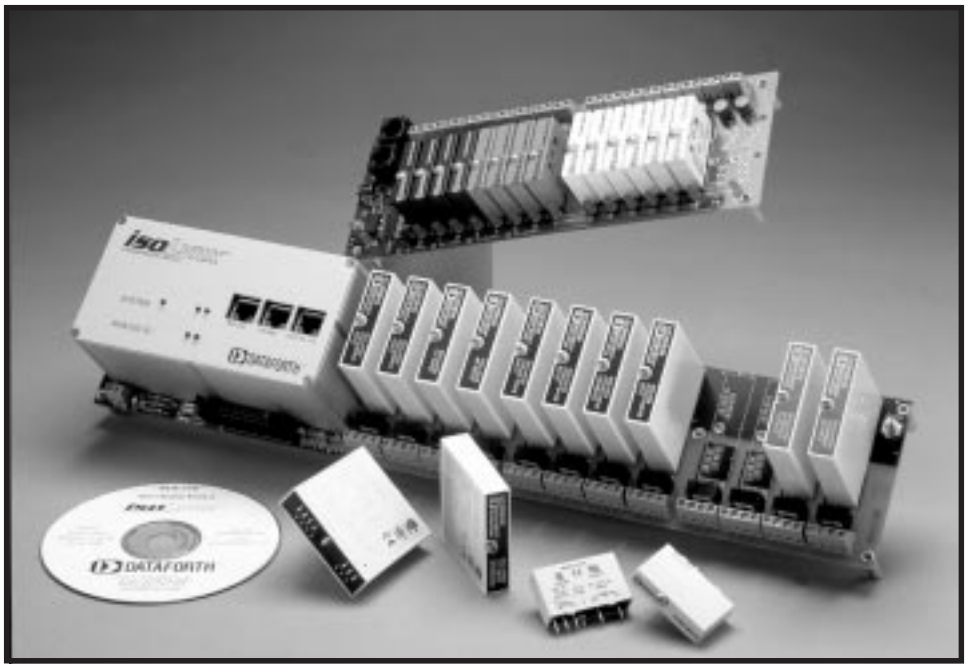
isoLynx™

Software User Manual



DATAFORTH®

isoLynx™ Software User Manual

isoLynx™ Software User Manual
MA1018 – 11-05-02

The information in this manual has been checked carefully and is believed to be accurate; however, Dataforth assumes no responsibility for possible inaccuracies or omissions. Specifications are subject to change without notice.

isoLynx is a trademark of Dataforth Corporation.
IBM, IBM PC, XT, AT, and PS/2 are trademarks or registered trademarks of International Business Machines Corporation.
Microsoft, MS-DOS, Windows, Windows NT, Visual Studio, Visual C++, and Visual BASIC are trademarks or registered trademarks of Microsoft Corporation.

isoLynx™ Software User Manual 1

1 Introduction 5

2 isoLynx™ Command Reference 7

2.1 Command Reference Introduction 7
2.1.1 Overview 7
2.1.2 System Configuration 8

2.2 Programming 11
2.2.1 Overview 11
2.2.2 Carrying Out Message Transactions 15
2.2.3 isoLynx™ Backpanel Jumper Settings 17

2.3 Technical Information 18
2.3.1 Analog I/O Units 18
2.3.2 Digital I/O Units 19
2.3.3 System Throughput 21

2.4 Command Directory 22
2.4.1 Command and Response Structure 22
2.4.2 Command Summary 25

2.5 Setup/System Commands 26
2.5.1 Read isoLynx Status Command ? 26
2.5.2 Reset Command B 29
2.5.3 Reset Parameters to Default Command [..... 31
2.5.4 Set System Parameters Command @ 33

2.6 I/O Configuration Commands 36
2.6.1 Read I/O Configuration - Group Command Y 36
2.6.2 Set I/O Configuration – Group Command G 39

2.7 Input Commands 42
2.7.1 Read Analog or Digital Default Output Values - Group Command * 42
2.7.2 Read Analog or Digital Inputs - Group Command R 45
2.7.3 Read Analog or Digital Input Command r 48
2.7.4 Read Averaging Sample Weight Command (..... 51

2.8 Output Commands 53
2.8.1 Set Analog or Digital Default Output Values - Group Command & 53
2.8.2 Set Analog or Digital Outputs - Group Command X 56
2.8.3 Set Analog or Digital Output Command x 59
2.8.4 Set Averaging Sample Weight Command h 62

2.9 Network Commands 64
2.9.1 Read Ethernet Configuration Command + ... 64
2.9.2 Set Ethernet Configuration Command # 67

2.10 Usage Examples 69
2.10.1 Configuring and Running Input and Output Channels 69
2.10.2 Changing from RS-232 to Ethernet and back 70
2.10.3 Changing from RS-232 to RS-485 2-wire and back 73
2.10.4 Changing from RS-485 2-wire to Ethernet and back 76
2.10.5 Changing from RS-485 2-wire to 4-wire and back 79

3 isoLynx Data Acquisition Library 82

3.1 Library Organization 82

3.2 Compiling, Linking, and Executing User Applications....82
3.2.1 Compiling/Linking with Visual C++....82
3.2.2 Compiling/Linking with Visual Basic....83
3.2.3 Executing Applications....83

3.3 isoLynx API....84
3.3.1 Error Codes....84
3.3.2 Library Functions and Data Structures....85
3.3.2.1 dLibClose....85
3.3.2.2 dLibError....86
3.3.2.3 dLibLog....88
3.3.2.4 dLibOpen....89
3.3.2.5 LIB_OPEN_STRUCT....90
3.3.3 Communications Functions and Data Structures....93
3.3.3.1 dComClose....93
3.3.3.2 dComConfigure....94
3.3.3.3 dComInquire....96
3.3.3.4 dComOpen....98
3.3.3.5 dComReceive....100
3.3.3.6 dComSend....102
3.3.3.7 dComStatus....104
3.3.3.8 COMCONF_SERIAL_STRUCT....106
3.3.3.9 COMCONF_SOCKET_STRUCT....107
3.3.3.10 COMOPEN_SERIAL_STRUCT....108
3.3.3.11 COMOPEN_SOCKET_STRUCT....113
3.3.4 IO Functions and Data Structures....117
3.3.4.1 dIoClose....117
3.3.4.2 dIoConfigure....118
3.3.4.3 dIoInquire....120
3.3.4.4 dIoOpen....122
3.3.4.5 dIoRead....123
3.3.4.6 dIoReset....125
3.3.4.7 dIoWrite....127
3.3.4.8 IOATTR_CHANNEL_CNF_STRUCT....129
3.3.4.9 IOATTR_ISOLYNX_AIOPTION_STRUCT....131
3.3.4.10 IOATTR_ISOLYNX_AOOPTION_STRUCT....133
3.3.4.11 IOATTR_ISOLYNX_DIOPTION_STRUCT....135
3.3.4.12 IOATTR_ISOLYNX_DOOPTION_STRUCT....137
3.3.4.13 IOATTR_ISOLYNX_INTERFACE_STRUCT....139
3.3.4.14 IOATTR_ISOLYNX_NETOPTION_STRUCT....142
3.3.4.15 IOCTRL_DEFAULT_STRUCT....144
3.3.4.16 IOOPEN_ISOLYNX_STRUCT....147
3.3.5 File Functions....150
3.3.5.1 dFileReadF....150
3.3.5.2 dFileReadI....152
3.3.5.3 dFileReadS....154
3.3.5.4 dFileWriteF....156
3.3.5.5 dFileWriteI....158
3.3.5.6 dFileWriteS....160

3.4 Script Files....162

3.5 Sample Applications....164
3.5.1 Running the Samples....164
3.5.2 Establishing a Connection....164
3.5.3 Configuration Sample....166
3.5.4 Input Sample....168

3.5.5 Output Sample....170

4 Utilities 172

4.1 LabVIEW....172
4.1.1 LabVIEW Virtual Instruments....173
4.1.1.1 Open Library.VI....173
4.1.1.2 Open isoLynx Dev.VI....174
4.1.1.3 Read Data.VI....176
4.1.1.4 Write Data.VI....178
4.1.1.5 Close isoLynx Dev.VI....180
4.1.1.6 Close Library.VI....181
4.1.1.7 Decode Error.VI....182
4.1.2 Sample LabVIEW Applications....183
4.1.2.1 Running the LabVIEW Samples....183
4.1.3 Creating LabVIEW Applications....184

Appendix A Troubleshooting Guidelines 185

A.1 isoLynx™ Controller "A/D" LED Blink Patterns 185
A.2 If the isoLynx™ Does Not Communicate or Sends Garbled Data From Any Interface 186
A.3 Selected Error Codes and Some of Their Less Obvious Causes 187
A.4 Unexpected Data Values and Some Possible Causes 187

Appendix B ASCII Character Table 188

Appendix C Warranty, Disclaimers, Return/Repair Policy 189

1 Introduction

Overview

An isoLynx™ Industrial Data Acquisition System can have a PC or other type of computer controlling from one to 16 intelligent isoLynx™ Analog I/O Base Units and up to 128 isoLynx™ Digital I/O Backpanels. The I/O units respond to a software protocol that we call isoLynx™ Command Protocol.

This manual describes the communication protocol required to communicate with isoLynx™ I/O units. Every isoLynx™ Command Protocol command is fully explained and illustrated by a programming example. Detailed explanations of the functions each command performs will be helpful to users of our isoLynx™ DAQ Library.

While it is possible to create code to talk directly to the isoLynx™ I/O units, the use of a high-level language and our isoLynx™ DAQ Library is encouraged for most applications.

About This Manual

This manual is organized as follows:

- Chapter 1: Introduction — Features and Configuration Information
- Chapter 2: isoLynx™ Command Reference
- Chapter 3: isoLynx™ DAQ Library
- Chapter 4: Utilities

Related Documents

The following documents contain additional information:

isoLynx™ Quick Start Guide
isoLynx™ Hardware User Manual

About Dataforth Corporation

“Our passion at Dataforth Corporation is designing, manufacturing, and marketing the best possible signal conditioning and data communication products. Our mission is setting new standards of product quality, performance, and customer service.” Dataforth Corporation, with over 17 years experience, is the worldwide leader in Instrument Class™ Industrial Electronics – rugged, high performance signal conditioning and data communication products that play a vital role in maintaining the integrity of industrial automation, data acquisition, and quality assurance systems. Our products directly connect to most industrial sensors and protect valuable measurement and control signals

and equipment from the dangerous and degrading effects of noise, transient power surges, internal ground loops, and other hazards present in industrial environments.

Dataforth spans the globe with over 50 International Distributors and US Representative Companies. Our customers benefit from a team of over 130 sales people highly trained in the application of precision products for industrial markets. In addition, we have a team of application engineers in our Tucson factory ready to address and solve any in-depth application questions. Upon receipt of a quote or order, our Customer Service Department provides fast one-day response of delivery information. We maintain an ample inventory that allows small quantity orders to be shipped from stock.

Contacting Dataforth Corporation

Contact Method	**Contact Information**
E-Mail:	
Technical Support:	techinfo@dataforth.com
Website:	www.dataforth.com
Phone:	520 741 1404 and 800 444 7644
Fax:	520 741 0762
Mail:	Dataforth Corporation 3331 E. Hemisphere Loop Tucson, AZ 85706

Errata Sheets

Refer to the Technical Support area of Dataforth's web site (www.dataforth.com) for any errata information on this product.

2 isoLynx™ Command Reference

2.1 Command Reference Introduction

2.1.1 Overview

2.1.1.1 Scope of Document

This manual describes the format of the data required to communicate with the family of isoLynx I/O (input/output) units using the isoLynx Command Protocol. The user of this manual may either be analyzing the data created by an isoLynx I/O unit in an effort to optimize the use of the high-level commands, or will be creating the data in some other program written in a high-level language. Data is exactly the same whether created by an isoLynx I/O unit, a PC using our isoLynx DAQ Library, or from custom-designed software.

Data link hardware topology, overall system hardware design, and the layout and connection of hardware pieces are beyond the scope of this document. Please refer to the isoLynx™ Hardware User Manual or the individual data sheets for specific analog and digital I/O modules for this level of detail.

2.1.1.2 isoLynx™ System Overview

The Dataforth isoLynx System is a family of intelligent digital and analog input/output controllers that operate as slave devices to a host computer. Physically, each isoLynx System is a modular assembly that consists of an isoLynx Analog I/O Base Unit with an isoLynx controller and screw terminal blocks, a processor board with a microcontroller, an I/O signal converter board, an industrial communication board, and high-density, isolated analog I/O modules or additionally, an isoLynx Digital I/O Backpanel with screw terminal blocks, a high speed microcontroller, and isolated digital I/O modules.

The removable processor board contains a microcontroller that communicates with a host computer equipped with the proper interface card. The processor board controls the plug-in I/O modules located on the Analog I/O Base Unit Backpanel and/or Analog I/O Expansion Backpanels and/or Digital I/O Backpanels.

Any combination of analog I/O modules may be plugged into an Analog I/O Base Unit and likewise, any combination of digital I/O modules may be plugged into a Digital I/O Backpanel. isoLynx analog and digital I/O units can then be combined on the same serial data link providing endless combinations of analog and digital I/O points.

You may use any one of a number of network/electrical interfaces with isoLynx I/O units in an installation.

* The RS-232 data link offers quick system setup and check out.

* The RS-485 data link offers excellent noise immunity and long cable lengths and multidrop operation over as few as a single twisted pair cable.

* Ethernet offers high speed networking, a common industrial protocol, and isolated communications.

2.1.2 System Configuration

2.1.2.1 Physical Layout

The high cost of electrical wiring and the noise susceptibility of analog signals make it desirable to place the control and monitoring points as close to the industrial process as possible. isoLynx I/O units offer design flexibility, with as few as one point or as many as 960 (16 x 60) analog I/O and 2048 (128 x 16) digital I/O points in one physical location.

RS-232 is limited to point-to-point communications links. In other words, for each isoLynx I/O unit you have chosen to use RS-232, you must have a corresponding RS-232 port on the host computer.

RS-485 and Ethernet choices are capable of multi-point networking. In laying out an installation, plan to route the data link cable to all points where you need to install remote I/O units now, and to all points where you may need data acquisition/control in the future.

Make provisions to supply DC power to each I/O unit location, preferably with a local power supply.

Assign a unique address (0 – 15 (0-F Hex)) to each isoLynx I/O unit. Addresses may appear in any order from beginning to end of the data link, however every address must be unique — no two controllers may share the same address. There is no requirement for consecutive addresses; the entire range of addresses may be used. Try to relate the address to a location or function.

2.1.2.2 Communications

The RS-232 data link offers quick system setup and check out. RS-232 is limited to point-to-point communications links. In other words, for each isoLynx I/O unit you have chosen to use RS-232, you must have a corresponding RS-232 port on the host computer.

The RS-485 data link offers excellent noise immunity, long cable lengths, and multidrop operation. The maximum cable length for isoLynx I/O units is 4,000 feet. In a multidrop configuration, up to 16 I/O units can be connected to a single serial port. In this configuration, all of the isoLynx I/O units are wired in parallel on the same serial data link. This provides security against power failures at individual I/O unit sites. If power is lost on an isoLynx I/O unit, communications to that board are lost; but communication to the rest of the isoLynx I/O units on the link remains intact.

Ethernet offers high speed networking, a common industrial protocol, and isolated communications. Each Ethernet link can be up to 250 feet long, but with multiple host computers and hubs in the network, the distance can be extended into the thousands of feet. Since Ethernet protocol encapsulates the isoLynx Command Protocol all of the security and reliability features of Ethernet are imparted to the networked isoLynx's.

2.1.2.3 Modular Construction

Each isoLynx I/O unit is composed of an Analog I/O base unit with an isoLynx controller and screw terminal blocks, a processor board with a microcontroller, an I/O signal converter board, an industrial communication board, and high-density, isolated analog I/O modules or a Digital I/O Backpanel with screw terminal blocks, a high speed microcontroller, and isolated digital I/O modules. Please refer to the isoLynx™ Hardware User Manual and individual I/O module data sheets for specific construction and connection details.

2.1.2.4 Serial Data Transmission

Remote I/O units are connected to the host computer via a twisted-pairs cable. One or two pairs carry command/response data depending on the choice of network electrical interface. Remote I/O units support the common data rates (1,200, 2,400, 4,800, 9,600, 19.2K, 38.4K, 57.6K, 76.8K, and 115.2K bits per second (bps)(Baud)) which are selectable via the Set System Parameters command. Selection of data rate often depends upon the capabilities of the host port. Many host computers have a limited data rate. Modems and radio links often operate at 1,200 or 2,400 bits per second (bps)(Baud). System throughput is increased by using the fastest available data rate.

2.1.2.5 Protocol

All isoLynx I/O units use the same command set and simple ASCII code is supported. ASCII code enables viewing the commands and responses on an ASCII display terminal, usually an LCD or CRT type. The isoLynx Command Protocol requires the transmission of two messages on the serial link every time a command is executed. The host sends a command to an I/O unit and then receives a response message acknowledging successful execution of the command along with any requested data, or an error message indicating that the I/O unit detected an error in the command message and was unable to execute it. As a consequence, the isoLynx Command Protocol is inherently half duplex.

2.1.2.6 Data Verification

To ensure secure data transmission, every command message and every response from an isoLynx I/O unit includes a data verification field. The data verification method is checksum modulo 256 (8-bit). Two checksum characters are sent and received in ASCII code.

2.2 Programming

2.2.1 Overview

The isoLynx I/O unit is an intelligent device that acts as a slave to a host computer. The host computer issues instructions to the isoLynx I/O unit by sending command message frames over a communications link. The addressed isoLynx I/O unit responds to the host by sending message frames back.

isoLynx I/O units are connected to the host computer via a twisted-pairs cable. Maximum data rate is 115K bits per second (bps) (Baud).

All isoLynx I/O units use the same command set and simple ASCII code is supported. ASCII code enables viewing the commands and responses on an ASCII display terminal, usually an LCD or CRT type. The isoLynx Command Protocol requires the transmission of two messages on the serial link every time a command is executed. The host sends a command to an I/O unit and then receives a response message acknowledging successful execution of the command along with any requested data, or an error message indicating that the I/O unit detected an error in the command message and was unable to execute it.

To ensure secure data transmission, every command message and every response from an isoLynx I/O unit includes a data verification field. The data verification method is checksum modulo 256 (8-bit). Two checksum characters are sent and received in ASCII code.

The host computer calculates the data verification field data (checksum) and sends it along as part of the command message to an isoLynx I/O unit. When an isoLynx I/O unit receives the message, it calculates its own data verification field value and compares that value with the transmitted checksum. If they match, the isoLynx I/O unit can verify that the message was received correctly.

The same procedure is repeated whenever an isoLynx I/O unit returns data to the host computer. When a message is received from an isoLynx I/O unit, the host computer calculates the data verification field data (checksum) and compares it against the data verification field that was transmitted as part of the message.

2.2.1.1 Error Code Summary – Error Messages

isoLynx is capable of detecting and reporting errors to the host computer. It always returns an acknowledgment message after successfully executing a command. If isoLynx detects an error, it will not execute the command, but will return an error message containing a code describing the exact nature of the error.

The error codes are:

01 Undefined Command
The command character was not a legal command character.

02 Checksum Error
The checksum received did not match the value calculated from the command.

03 Receive Overrun Error
The command exceeded the receive buffer limit of 80 characters.
The receive registers were overrun probably due to a data rate mismatch. Data rate mismatch must be +/-4.0%.

04 Reserved.

05 Data Field Error
Address specified not in bounds or wrong number of characters received.

06 Communications link Watchdog Time-Out Error
The Host communications link watchdog timer has expired.
Possible causes: the Host was hung or rebooted, or comm cable was disconnected or is intermittent.
The Digital I/O communications link watchdog timer has expired.
Possible causes: the addressed Digital I/O Panel is not present, not powered, or faulty or comm cable not connected or faulty.

07 Specified Data Invalid Error
One or more data fields contain an illegal value, decimal 0-9 or hexadecimal A-F expected.

08 Reserved.

09 Invalid Module Type Error
This code is returned when one or more specified modules is not of the type requested by the command.
Possible causes: attempting read from an output or write to an input or read or write to a vacant channel.

10 Reserved.

11 Reserved.

12 EEPROM write error
Each EEPROM write is verified; one or more verifications failed. You may choose to continue data acquisition or to call your regional Dataforth representative or Dataforth directly at 800 444 7644.

13 Invalid Panel Type
One to all module (M) bits for channels 15-12, were set for panel 0 for Set Average Weight command (h), Set I/O Configuration - Group command (G), Read I/O Configuration - Group command (Y), Read Input commands (R or r), or Write Output commands (X or x).
Or panel requested for any network command was not panel 0.
Or panels 4-7 were requested for any command.

14 I/O Configuration Type Error
The number of type fields does not match the number of module (M) bits in Set I/O Configuration - Group command (G).

15 I/O Configuration Missing
No channels in this panel have been configured or some to all channels requested have not been configured or are not present. This error code could be returned after a Read Input command (R or r) or a Write Output command (X or x).

16 Panel Data Rate Error
Requested data rate is out of range for isoLynx panel type.

17 Invalid Requested Data Type
Requested data type is undefined.

18 A/D Busy
The A/D converter is not functioning correctly or at all. This is a hardware problem returned as an error message to a Read Input command (R or r) to alert the user. You may choose to continue data acquisition or to call your regional Dataforth representative or Dataforth directly at 800 444 7644.

2.2.1.2 Implementing an Application

To implement an application, the user must do the following:

(1) Understand how to communicate with an isoLynx I/O unit.
This will require:

- The ability to build command messages.
- The ability to carry out message transactions.
- The ability to interpret data returned by an isoLynx I/O unit.

(2) Understand how to manage an isoLynx I/O unit network.

This will require the ability to initialize and configure each isoLynx I/O unit in the network.

2.2.1.3 Building Command Messages

The structure of all isoLynx Command Protocol command messages (frames) is shown in the reference sections of this manual, Sections 2.5 through 2.9. Use this as a guide for building commands.

2.2.1.4 Numbers Representation in ASCII Code

The ASCII code for the isoLynx I/O unit uses the Hexadecimal (Hex) numbering system to represent numerical data in commands and responses. Appendix B contains an ASCII character table.

ASCII code means that command messages are transmitted as a series of ASCII characters. Numbers are transmitted as the ASCII number characters 0 through 9 and the upper case ASCII characters A through F. We will refer to an ASCII character representing a Hex number as “ASCII-Hex.”

Example:

15 (decimal) = F (Hex) and is transmitted as the ASCII “F” character.

2.2.1.5 8-Bit Checksum

For ASCII code, the 8-bit checksum is computed by adding the hexadecimal values of all the ASCII characters in the command frame (string), excluding the start of command character ">", the DVF (Data Verification Field), and the CR (Carriage Return) character at the end. For an 8-bit checksum, put the ASCII equivalent of the two least significant Hex digits into the checksum position in the command string just preceding the Carriage Return (CR) character.

Example:

>A1x0A3CD045CR is a valid command.
A1x0A3CD0 is the part used to calculate the checksum.

Note: *The Start of Command character ">" is NOT part of the checksum calculation.*

The checksum is calculated by summing the Hexadecimal values of the ASCII characters which make up the command.

ASCII characters:	A	1	x	0	A	3	C	D	0
Hexadecimal values of characters:	41	31	78	30	41	33	43	44	30

41+31+78+30+41+33+43+44+30 = 0245 Hexadecimal sum
The two least significant digits are 45 Hexadecimal.
The checksum (45 Hex) is appended to the end of the command string. The complete command then becomes ">A1x0A3CD045CR". This command is also used as an example in the "Set Analog or Digital Output" command description in Section 2.8.3.

2.2.2 Carrying Out Message Transactions

The details of how to actually write the command message out to a host adapter card (RS-232 or RS-485) in the host computer will be found in the manual that comes with your host adapter card. The method will be the same as sending characters out of a COM1 or COM2 serial port on your host computer. Receiving the response messages is the same as receiving characters on your COM1 or COM2 serial port.

You must set the communications parameters of the host serial port to match the settings of the isoLynx serial port. For first time power-up and jumper reset of communication parameters to default, the factory default settings of RS-232 and 9.6 K bits per second (bps)(Baud) will be in effect. For any other reset, the most recent interface and data rate settings stored in EEPROM will be in effect. These settings should be documented in separate data blocks in a .dat file or in separate .dat files for each isoLynx in a network.

For ASCII code, you would have 1 start bit, 8 data bits, 1 stop bit, and no parity. There is nothing special about data transmission in ASCII code.

Dataforth supplies software which will take care of all the details of communicating to I/O units. It will do all the error checking, initialization, building the command strings, etc. If you are writing your own driver, you will need to do all this yourself. To help in developing your driver, you will want to use a terminal program, such as Hyperterminal, to manually send commands to the isoLynx and receiving responses from the isoLynx. Set local echo in your terminal program. Since the isoLynx does not send a line feed with a carriage return, you may want to set "append a line feed after each line" in the receive parameters in your terminal program to make the transactions more readable. This works for most serial interfaces, such as RS-232 and RS-485 4-wire. One exception is RS-485 2-wire with echo. In this case, set the terminal program to not send line feeds with carriage returns and set no local echo.

2.2.2.1 Interpreting the Response

The isoLynx I/O unit will respond to each command that contains its address in the isoLynx data field i. The structure of the response will depend upon the command given. Sections 2.5 through 2.9 of this manual give a complete description of the response to each command. If the command was not executed for some reason, the response will contain an error code. The meaning of these error codes is described in Section 2.2.1.1 of this manual. Each response message frame has a field which can be examined to determine if the command was executed or if there was an error. For ASCII protocol, the first character in the response is an A(cknowledged) if the command was successful or is an N(ot acknowledged) if there was an error. The error code is given following the N.

Examples of commands and responses are given for every command in Sections 2.5 through 2.9 of this manual.

2.2.2.2 Interpreting Analog Data

This section applies only to analog I/O units.

All analog input and output values are exchanged between the host computer and an isoLynx I/O unit in 16-bit data fields, the data is given in counts (raw data), giving a range from –32768 to +32767 counts. Analog input commands may also request averaged data.

2.2.2.3 Initializing an isoLynx™ Network

Many of the operating characteristics of isoLynx Analog and Digital I/O units can be selected by the host computer. On power up or after a RESET command, isoLynx Analog and Digital I/O units initialize themselves to a set of default characteristics stored in EEPROM. The factory default settings are shown in section 2.3.1.3 Analog I/O Factory Default Settings, in section 2.3.2.3 Digital I/O Factory Default Settings, and also in the various command descriptions.

The host computer is responsible for sending each isoLynx I/O unit in the network all of the commands necessary to select the desired characteristics.

For analog I/O units, initialization usually involves sending any setup or configuration command needed to configure your modules as inputs or outputs. Optionally, you may also send sampling average weights and default output values. Many of the parameters will be saved in EEPROM by the configuration commands and will be automatically restored upon power up.

For digital I/O units, initialization usually involves sending any setup or configuration command needed to configure your modules as inputs or outputs. Optionally, you may also send default output values. Many of the parameters will be saved in EEPROM by the configuration commands and will be automatically restored upon power up.

Note: *Since Dataforth's Analog and Digital I/O modules cannot electronically identify their functions, the isoLynx Analog I/O Base Unit and the isoLynx Digital I/O Backpanel cannot automatically update their I/O configuration data bases upon a change of module types in the system. When changing module types, the user must issue a Set I/O Configuration command so the host computer software, the isoLynx Analog I/O Base Unit and, if used, the isoLynx Digital I/O Backpanel will update their I/O configuration data bases.*

2.2.3 isoLynx™ Backpanel Jumper Settings

Please refer to the isoLynx™ Hardware User Manual for hardware configurations and jumper settings.

2.3 Technical Information

2.3.1 Analog I/O Units

2.3.1.1 Analog I/O Timing

Analog I/O units constantly update the status of their I/O. Input modules are read every 0.5 millisecond and the data is held in memory until requested by the host CPU. Output module data is held in memory and output to each module every 20 milliseconds.

Note: *The A/D system sampling rate is 0.5 millisecond. Since there is one A/D in the system, the input channel sample rate is variable over the number of input channels configured. One configured input channel is sampled once every 0.5 millisecond. For 2 configured input channels, each channel is sampled once every 2 x 0.5 millisecond or once every 1.0 millisecond. In general for n configured input channels, the sampling rate is n x 0.5 milliseconds per configured input channel.*

2.3.1.2 Input Averaging

An input channel may be instructed to average its input readings. The computations are done every 0.5 millisecond. The algorithm used is an incremental running average. The equation being solved is:

New Average = ((Current Reading - Old Average) / Weight) + Old Average

Weight ranges from one to 2^{14} in discrete powers of 2 increments.

Weight = 0, 2^0 to 2^{14}

Weights are entered into the Set Averaging Sample Weight command as an ASCII representation of a 4 digit Hexadecimal number, for example:

Weight = DDDD = 0020

The above example has a one in bit position 5. This translates to a decimal weight of 2^5 = 32.

A weight of zero signals the isoLynx Analog I/O Base Unit to stop averaging and to hold the last average in RAM. A weight of one is the same as current data.

Note: *The A/D system calculates an average once every 0.5 millisecond. Since there is one A/D in the system, the averaging calculation rate is variable over the number of input channels configured. For one configured input channel an average is*

calculated once every 0.5 millisecond. For 2 configured input channels, an average is calculated for each channel once every 2 x 0.5 millisecond or once every 1.0 millisecond. In general for n configured input channels, an average is calculated once every n x 0.5 milliseconds per configured input channel.

2.3.1.3 Analog I/O Factory Default Settings

For an isoLynx Analog I/O Base Unit fresh from the factory and/or after a Reset To Default command executes the following default conditions are set:

* All channels are set to vacant and to be inputs.
* All averages are set to zero.
* All average weights are set to zero.
* All default output values are set to zero.
* Ethernet Parameters are set to the following:

IP address	=	192.168.0.0	(C0.A8.00.00 Hex)
subnetmask	=	255.255.255.0	(FF.FF.FF.00 Hex)
gateway	=	192.168.0.0	(C0.A8.00.00 Hex)
dnsserver	=	100.100.100.100	(64.64.64.64 Hex)

In addition to the above conditions, an isoLynx Analog I/O Base Unit fresh from the factory has the following default conditions set:

* Communications interface is RS-232.
* Communications data rate is 9.6K bits per second (bps) (Baud).

2.3.2 Digital I/O Units

2.3.2.1 Digital I/O Timing

All Digital I/O units will read the I/O status or set the output state at the time the command is issued. When a “Read Digital Input” command is received, the digital I/O unit will read the current status then transmit the response data. Upon receipt of a “Set Digital Output” command, the digital I/O unit will affect the new output status before the acknowledge response is sent.

2.3.2.2 Digital I/O Backpanels As Standalone Units

isoLynx Digital I/O Backpanels may be operated as standalone units connected directly to a host computer. The built-in communications interface is RS-485 2-wire. If you desire Ethernet connectivity, there are a number of manufacturers offering RS-485 to Ethernet converters. For standalone hardware connections, reference the isoLynx™ Hardware User Manual.

isoLynx Command Protocol digital I/O compatible commands all have independent isoLynx and panel address parameters which lend themselves easily to Digital I/O standalone operation of the command protocol. For details on building the specific commands, reference sections 2.5 through 2.9 .

2.3.2.3 Digital I/O Factory Default Settings

For an isoLynx Digital I/O Backpanel fresh from the factory and/or after a Reset To Default command executes the following default conditions are set:

* All channels are set to vacant and to be inputs.
* All default output values are logic zero.

In addition to the above conditions, an isoLynx Digital I/O Backpanel fresh from the factory has the following default conditions set:

* Communications interface is RS-485 2-wire. This is the only interface setting available on the Digital I/O Backpanel since it is designed into the hardware.
* Communications data rate is 115.2K bits per second (bps) (Baud).

2.3.3 System Throughput

In most applications, system throughput is greatly affected by the rate at which I/O can be updated. For intelligent I/O units this involves sending a command string, processing of the command, then an acknowledge and/or data being returned. The time required for any transaction will be equal to the transmission time of all data in the command and response, plus the time required by the I/O unit to process the command.

Time Required to Read n Input Positions (Typical, using Read Input-Group Command)

I/0 Unit	n	Communication/ Data Transfer	Command Execution	Total
Analog	1	2.08 msec	0.25 msec	2.33 msec
	12	5.90 msec	0.51 msec	6.41 msec
	16	7.29 msec	0.65 msec	7.94 msec
Digital	1	3.3 msec	0.3 msec	3.6 msec
	16	3.5 msec	0.7 msec	4.2 msec
Digital (Stand Alone)	1	1.7 msec	0.3 msec	2.0 msec
	16	1.7 msec	0.7 msec	2.4 msec
All serial communication at 115.2K bits per second (Baud).				

Time Required to Write n Output Positions (Typical, using Write Output-Group Command)

I/0 Unit	n	Communication/ Data Transfer	Command Execution	Total
Analog	1	2.08 msec	0.50 msec	2.58 msec
	12	5.90 msec	6.50 msec	12.40 msec
	16	7.29 msec	8.50 msec	15.79 msec
Digital	1	3.3 msec	0.5 msec	3.8 msec
	16	4.2 msec	8.5 msec	12.7 msec
Digital (Stand Alone)	1	1.7 msec	0.5 msec	2.2 msec
	16	2.1 msec	8.5 msec	10.6 msec
All serial communication at 115.2K bits per second (Baud).				

2.4 Command Directory

2.4.1 Command and Response Structure

Command Name **COMMAND CMD**

The top line shows the NAME of the command at the left of the line and the command character CMD at the right following the word COMMAND. CMD is always a single ASCII character.

Description

Tells briefly what the command does.

Versions

Indicates whether the command is valid for Analog and/or Digital.

ASCII Command Format

Shows the correct format for the command message frame in ASCII Code.

ASCII is used because it is desirable to have all the characters in the command string be seen on a standard computer terminal connected to the communications line. The computer terminal must be equipped with the proper hardware interface (i.e. RS-485, RS-232 etc.). All characters in the string will be displayed on the terminal except for the Ending Delimiter character which is a carriage return. The computer terminal should have the auto line feed feature enabled so that each carriage return will start a new line on the computer terminal display screen.

All the fields that make up the command message frame will be shown in the following general format. Command Name COMMAND CMD.

# characters	1	1	1	1	0 or more	2	1
	SD	i	P	CMD	CDF	DVF	ED

Where:

SD	=	Starting Delimiter
i	=	isoLynx Address
P	=	Panel Address
CMD	=	Command Character
CDF	=	Command Data Fields
DVF	=	Data Verification Field
ED	=	Ending Delimiter

The Starting Delimiter will always be the ASCII character > (3E Hex). The Ending Delimiter is always an ASCII carriage return (0D Hex).

The Address fields are a character each. The i field contains one ASCII character that represents an address (0 to F Hex) for the isoLynx Address. The P field contains one ASCII character that represents an address (0 to F Hex) for the Panel Address. For example, if the address of the isoLynx is 11 (B Hex) and the address of the Panel is 15 (F Hex), then the two characters transmitted would be an ASCII B (42 Hex) and an ASCII F (46 Hex). The ASCII character B is transmitted first.

Note: *Document very carefully your isoLynx addresses in a network and keep the information current. Each isoLynx looks for its own address and ignores any commands not addressed to it. So for any commands which are addressing vacant address space, there is a potential for recurrent system time-outs*

The CMD field will be one ASCII character, the command character. The name of the command and the command characters are shown on the top line.

Some commands require that additional Command Data Fields (CDF) be transmitted. These Command Data Fields provide the additional information needed by the isoLynx Analog I/O Base Unit or the Digital I/O Backpanel before it can execute the command. For example, the SET OUTPUT command requires a Command Data Field that tells the Analog or Digital I/O units which output channel to set and the value to which to set it. Some commands do not require any additional data fields while other more powerful commands require many data fields. The command description sheet will show the format of each Command Data Field (CDF).

The Data Verification Field (DVF) requires two characters for the 8-bit checksum. See Section 2.2.1.5 for an explanation of how these characters can be constructed by the host computer for transmission to the I/O unit.

ASCII Response Format

Shows the format of the response message frame in ASCII code.

The response message frame is sent from the I/O unit to the host in response to the above ASCII command.

The fields that make up the response message frame will be shown in two general formats. The first shows the format of the response message frame transmitted by the addressed I/O unit for a successful completion of the command that was sent by the host. The second shows the format of the response message frame transmitted by the addressed I/O unit when an error occurred.

Success Response Message Frame

# characters	1	1	1	1	0 or more	2	1
	A	i	P	CMD	RDF	DVF	CR

Where:

A	=	ASCII Character A (41 Hex)
i	=	isoLynx Address
P	=	Panel Address
CMD	=	Command Character
RDF	=	Response Data Fields
DVF	=	Data Verification Field
CR	=	ASCII Carriage Return (0D Hex)

Error Response Message Frame

# characters	1	1	1	1	2	2	1
	N	i	P	CMD	EC	DVF	CR

Where:

N	=	ASCII Character N (4E Hex)
i	=	isoLynx Address
P	=	Panel Address
CMD	=	Command Character
EC	=	Error Code
DVF	=	Data Verification Field
CR	=	ASCII Carriage Return (0D Hex)

The Response Data Fields (RDF) are used for commands that require data to be transmitted back to the host computer. The command description sheet will show the format of each Response Data Field (RDF) when return data is required. For example, the READ ANALOG INPUT - GROUP command requires 16-bit analog data to be returned to the host computer.

When the I/O unit encounters an error in executing a command, it places an Error Code (EC) in the EC field of the Error Response Message Frame. See Section 2.2.1.1 for the meanings of the Error Codes.

The Data Verification Field (DVF) field will have two characters representing the 8-bit checksum. The I/O unit will compute the checksum and place it in the Data Verification Field of the response message frame.

Examples

Show actual command messages to and responses from the I/O unit that demonstrate the use of the command.

2.4.2 Command Summary

The following summary lists the commands by related groups. The version indicates whether the command is an A(nalog) or D(igital) command.

Section 2.5: Setup / System Commands

Command Name	Command Format	Version
Read isoLynx™ Status	?	A,D
Reset	B	A,D
Reset Parameters To Default	[	A,D
Set System Parameters	@ DR NI CF	A,D

Section 2.6: I/O Configuration Commands

Command Name	Command Format	Version
Read I/O Configuration-Group	Y	A,D
Set I/O Configuration-Group	G MMMM TT	A,D

Section 2.7: Input Commands

Command Name	Command Format	Version
Read Analog or Digital Default Output Values – Group	* MMMM	A,D
Read Analog or Digital Inputs – Group	R MMMM TT	A,D
Read Analog or Digital Input	r CC TT	A,D
Read Averaging Sample Weight	(CC	A

Section 2.8: Output Commands

Command Name	Command Format	Version
Set Analog or Digital Default Output Values – Group	& MMMM DDDD	A,D
Set Analog or Digital Outputs – Group	X MMMM DDDD	A,D
Set Analog or Digital Output	x CC DDDD	A,D
Set Averaging Sample Weight	h CC DDDD	A

Section 2.9: Network Commands

Command Name	Command Format	Version
Read Ethernet Configuration	+	A
Set Ethernet Configuration	# IP sn gw dns dhcp res	A

2.5 Setup/System Commands

2.5.1 Read isoLynx Status Command ?

Description:

Reads isoLynx status bytes and responds. Status information includes firmware version, serial number, power-on self test results, and communications parameters.

Versions:

Analog and Digital

ASCII Command Format:

# characters	1	1	1	1	2	1
	>	i	P	?	DVF	CR

Where:

>	=	ASCII Character > (3E Hex)
i	=	isoLynx Address is: 0 – F
P	=	Panel Address is: 0 – F
		Analog Panel: 0 – 3
		Reserved: 4 – 7
		Digital Panel: 8 – F
?	=	Command Character (3F Hex)
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

ASCII Response Format:

Success Response Format

# characters	1	1	1	1	4	9	1	1	2	2	1
	A	i	P	?	V	SN	RE	NI	DR	DVF	CR

Where:

A	=	Good Response Character (41 Hex)(Ack)
i	=	isoLynx Address
P	=	Panel Address
?	=	Command Character (3F Hex)
V	=	Firmware Version (V1.0.0 V100)
SN	=	Serial Number-Date Code (NNNNNYYWW)
		NNNNN = Serial Number, YY = Year, WW = Week

RE = Power-On Self Test Results (Bn: Pass = 0, Fail = 1)
B3: Reserved, B2: EEPROM, B1: A/D, B0: D/A Analog I/O
B3: Reserved, B2: EEPROM, B1: Reserved, B0: Reserved Digital I/O

NI = Network Interface
0 = RS-232 (isoLynx Analog I/O Base Unit's factory default)
1 = RS-485, 2-wire (isoLynx Digital I/O Backpanel's only choice)
2 = RS-485, 4-wire
3 = Ethernet

DR = Data Rate in bits per second (bps) = Baud
01 = 115.2K bps (isoLynx Digital I/O Backpanel's factory default)
03 = 57.6K bps
05 = 38.4K bps
0B = 19.2K bps
17 = 9.6K bps (isoLynx Analog I/O Base Unit's factory default)
2F = 4.8K bps
5F = 2.4K bps
BF = 1.2K bps
EE = Ethernet

DVF = Data Verification Field

CR = Carriage Return (0D Hex)

Error Response Format

# characters	1	1	1	1	2	2	1
	N	i	P	?	EC	DVF	CR

Where:

N = Error Response Character (4E Hex)(Nack)

i = isoLynx Address

P = Panel Address

? = Command Character (3F Hex)

EC = Error Code (Reference Error Code Summary, Section 2.2.1.1)

DVF = Data Verification Field

CR = Carriage Return (0D Hex)

Examples:

Command	>A0 ? B1 CR
Response	A A0 ? V100 01234 02 30 0 2 0B 6B CR

Sends a Read isoLynx Status command in ASCII format to the isoLynx Analog I/O Base Unit at address A Hex. Five data fields, which have been split into seven for visual clarity, are returned. The firmware version number is V 1.0.0, shown as V100. The serial number is 01234, manufacturing year is 2002, shown as 02, and the manufacturing week is 30. The power-on self test results are EEPROM, A/D, and D/A all pass, shown as 0. The network interface is RS-485 4-wire, shown as 2. The data rate is 19.2K bps, shown as 0B.

Command	>A9 ? B1 CR
Response	A A9 ? V100 01212 02 30 0 1 0B 6F CR

Sends a Read isoLynx Status command in ASCII format to the isoLynx Digital I/O Panel 1 at address A Hex. Five data fields, which have been split into seven for visual clarity, are returned. The firmware version number is V 1.0.0, shown as V100. The serial number is 01212, manufacturing year is 2002, shown as 02, and the manufacturing week is 30. The power-on self test results are EEPROM passes, shown as 0. The network interface is RS-485 2-wire, shown as 1. The data rate is 115.2K bps, shown as 01.

NOTE: Since 8-F in the panel address indicates a digital panel, subtract 8 from the panel address to get the physical panel address.

Data verification method is 8-bit checksum.

Note: *Blank spaces in the above examples are added for visual clarity only. They do not actually appear in the command or in the response.*

2.5.2 Reset Command B

Description:

Forces hardware reset. This reset command restores the configuration stored in EEPROM since it is the primary storage for factory default and for subsequent user selected settings.

Versions:

Analog and Digital

ASCII Command Format:

# characters	1	1	1	1	2	1
	>	i	P	B	DVF	CR

Where:

> = ASCII Character > (3E Hex)
> i = isoLynx Address: 0 – F
> P = Panel Address: 0 – F
> Analog Panel: 0 – 3
> Reserved: 4 – 7
> Digital Panel: 8 – F
> B = Command Character (42 Hex)
> DVF = Data Verification Field
> CR = Carriage Return (0D Hex)

ASCII Response Format:

Success Response Format

# characters	1	1	1	1	2	1
	A	i	P	B	DVF	CR

Where:

> A = Good Response Character (41 Hex)(Ack)
> i = isoLynx Address
> P = Panel Address
> B = Command Character (42 Hex)
> DVF = Data Verification Field
> CR = Carriage Return (0D Hex)

Error Response Format

# characters	1	1	1	1	2	2	1
	N	i	P	B	EC	DVF	CR

Where:

N	=	Error Response Character (4E Hex)(Nack)
i	=	isoLynx Address
P	=	Panel Address
B	=	Command Character (42 Hex)
EC	=	Error Code (Reference Error Code Summary, Section 2.2.1.1)
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

Examples:

Command > A0 B B3 CR
Response A A0 B F4 CR

Sends a Reset command in ASCII format to the isoLynx Analog I/O Base Unit (Panel 0) at address A Hex. This reset command restores the configuration stored in EEPROM. It sends the reply after the reset has been completed.

Command > A9 B BC CR
Response A A9 B FD CR

Sends a Reset command in ASCII format to the isoLynx Digital I/O Panel 1 at address A Hex. This reset command restores the configuration stored in EEPROM. It sends the reply after the reset has been completed.

NOTE: Since 8-F in the panel address indicates a digital panel, subtract 8 from the panel address to get the physical panel address.

Data verification method is 8-bit checksum.

Note: *Blank spaces in the above examples are added for visual clarity only. They do not actually appear in the command or in the response.*

2.5.3 Reset Parameters to Default Command [

Description:

This command resets all but communications parameters to factory default values from ROM. Writes factory default values to EEPROM.

Versions:

Analog and Digital

ASCII Command Format:

# characters	1	1	1	1	2	1
	>	i	P	[	DVF	CR

Where:

>	=	ASCII Character > (3E Hex)
i	=	isoLynx Address: 0 – F
P	=	Panel Address: 0 – F
		Analog Panel: 0 – 3
		Reserved: 4 – 7
		Digital Panel: 8 – F
[	=	Command Character (5B Hex)
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

ASCII Response Format:

Success Response Format

# characters	1	1	1	1	2	1
	A	i	P	[	DVF	CR

Where:

A	=	Good Response Character (41 Hex)(Ack)
i	=	isoLynx Address
P	=	Panel Address
[	=	Command Character (5B Hex)
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

Error Response Format

# characters	1	1	1	1	2	2	1
	N	i	P	[	EC	DVF	CR

Where:

N	=	Error Response Character (4E Hex)(Nack)
i	=	isoLynx Address
P	=	Panel Address
[	=	Command Character (5B Hex)
EC	=	Error Code (Reference Error Code Summary, Section 2.2.1.1)
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

Examples:

Command	> A0 [CC CR
Response	A A0 [0D CR

Sends a Reset to Default command in ASCII format to the isoLynx Analog I/O Base Unit (Panel 0) at address A Hex. This reset command resets all but communications parameters to factory default values from ROM and writes factory default values to EEPROM. It sends the reply after the reset has been completed.

Command	> A9 [D5 CR
Response	A A9 [16 CR

Sends a Reset to Default command in ASCII format to the isoLynx Digital I/O Panel 1 at address A Hex. This reset command resets all but communications parameters to factory default values from ROM and writes factory default values to EEPROM. It sends the reply after the reset has been completed. It sends the reply after the reset has been completed.

NOTE: Since 8-F in the panel address indicates a digital panel, subtract 8 from the panel address to get the physical panel address.

Data verification method is 8-bit checksum.

Note: *Blank spaces in the above examples are added for visual clarity only. They do not actually appear in the command or in the response.*

2.5.4 Set System Parameters Command @

Description:

Set or clear the bits in the System Parameter Control Bytes which select system communication configuration. A detailed description of the Control Bytes is presented below. Changed Control Byte values are written to EEPROM when this command is executed and are restored upon system reset due to watchdog timeout, brown-out, power cycling, the issue of Reset command (B) or Reset to Default command ([ ). Factory default values are shown below.

Versions:

Analog and Digital

ASCII Command Format:

# characters	1	1	1	1	1	1	2	2	1
	>	i	P	@	NI	CF	DR	DVF	CR

Where:

> = ASCII Character > (3E Hex)
>
> i = isoLynx Address: 0 – F
>
> P = Panel Address: 0 – F
>
> Analog Panel: 0 – 3
>
> Reserved: 4 – 7
>
> Digital Panel: 8 – F
>
> @ = Command Character (40 Hex)
>
> NI = Network Interface (Analog Only)
>
> 0 = RS-232 (isoLynx Analog I/O Base Unit's factory default)
>
> 1 = RS-485, 2-wire (isoLynx Digital I/O Backpanel's only choice)
>
> 2 = RS-485, 4-wire
>
> 3 = Ethernet
>
> CF = Communications Configuration (Analog Only)
>
> 0 = RS-232 (isoLynx Analog I/O Base Unit's factory default)
>
> 1 = RS-485 point-to-point w/ echo
>
> 2 = RS-485 multi-drop w/ echo
>
> 3 = RS-485 point-to-point, no echo
>
> 4 = RS-485 multi-drop, no echo

DR = Data Rate in bits per second (bps) = Baud
01 = 115.2K bps (isoLynx Digital I/O Backpanel's factory default)
03 = 57.6K bps
05 = 38.4K bps
0B = 19.2K bps
17 = 9.6K bps (isoLynx Analog I/O Base Unit's factory default)
2F = 4.8K bps
5F = 2.4K bps
BF = 1.2K bps
DVF = Data Verification Field
CR = Carriage Return (0D Hex)

ASCII Response Format:

Success Response Format

# characters	1	1	1	1	2	1
	A	i	P	@	DVF	CR

Where:
A = Good Response Character (41 Hex)(Ack)
i = isoLynx Address
P = Panel Address
@ = Command Character (40 Hex)
DVF = Data Verification Field
CR = Carriage Return (0D Hex)

Error Response Format

# characters	1	1	1	1	2	2	1
	N	i	P	@	EC	DVF	CR

Where:
N = Error Response Character (4E Hex)(Nack)
i = isoLynx Address
P = Panel Address
@ = Command Character (40 Hex)
EC = Error Code (Reference Error Code Summary, Section 2.2.1.1)
DVF = Data Verification Field
CR = Carriage Return (0D Hex)

Examples:

Command	> A1 @ 2 4 0B 8A CR
Response	A A1 @ F3 CR

Sends a Set System Parameters command in ASCII format to the isoLynx Analog I/O Panel 1 at address A Hex. Network Interface is set to RS-485 4-wire, Communications Configuration is RS-485 multi-drop, no echo, and Data Rate is 19.2 Kbps (Baud).

Command	> A9 @ 1 2 2F 95 CR
Response	A A9 @ FB CR

Sends a Set System Parameters command in ASCII format to the isoLynx Digital I/O Panel 1 at address A Hex. Network Interface is set to RS-485 2-wire, Communications Configuration is RS-485 multi-drop with echo, and Data Rate is 4.8 Kbps (Baud).

NOTE: Since 8-F in the panel address indicates a digital panel, subtract 8 from the panel address to get the physical panel address.

Data verification method is 8-bit checksum.

Note: *Blank spaces in the above examples are added for visual clarity only. They do not actually appear in the command or in the response.*

2.6 I/O Configuration Commands

2.6.1 Read I/O Configuration - Group Command Y

Description:
This command causes the addressed panel to send back a response to the host that includes module presence and module type (I/O) for each of its 12 or 16 channels.

Versions:
Analog and Digital

ASCII Command Format:

# characters	1	1	1	1	2	1
	>	i	P	Y	DVF	CR

Where:

>	=	ASCII Character > (3E Hex)
i	=	isoLynx Address: 0 – F
P	=	Panel Address: 0 – F
		Analog Panel: 0 – 3
		Reserved: 4 – 7
		Digital Panel: 8 – F
Y	=	Command Character (59 Hex)
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

ASCII Response Format:

Success Response Format

# characters	1	1	1	1	4	2 to 32	2	1
	A	i	P	Y	MMMM	TT	DVF	CR

Where:

A	=	Good Response Character (41 Hex)(Ack)
i	=	isoLynx Address
P	=	Panel Address
Y	=	Command Character (59 Hex)
MMMM	=	Module Presence
TT	=	Module Type
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

Channel Binary Format MMMM (4 Character):
1 = Module Present
0 = Module Not Present

Data character	3	2	1	0
Module Presence	15 – 12	11 – 8	7 – 4	3 – 0
Example	1 0 0 0	0 0 0 0	1 1 1 1	0 0 0 1
Hex ASCII Char	8	0	F	1

Module type TT:

00 Hex Analog or Digital Input
80 Hex Analog or Digital Output
(Analog or Digital is specified above in Panel Address)

Error Response Format

# characters	1	1	1	1	2	2	1
	N	i	P	Y	EC	DVF	CR

Where:

N = Error Response Character (4E Hex)(Nack)
i = isoLynx Address
P = Panel Address
Y = Command Character (59 Hex)
EC = Error Code (Reference Error Code Summary, Section 2.2.1.1)
DVF = Data Verification Field
CR = Carriage Return (0D Hex)

Examples:

Command > A1 Y CB CR
Response A A1 Y 0A05 80 80 00 00 72 CR

Sends a Read I/O Configuration – Group command in ASCII format to the isoLynx Analog I/O Panel 1 at address A Hex. Channels 11 and 09 are configured as outputs, and channels 03 and 00 are configured as inputs.

Command > A9 Y D3 CR
Response A A9 Y 0A05 80 80 00 00 7A CR

Sends a Read I/O Configuration – Group command in ASCII format to the isoLynx Digital I/O Panel 1 at address A Hex. Channels 11 and 09 are configured as outputs, and channels 03 and 00 are configured as inputs.
NOTE: Since 8-F in the panel address indicates a digital panel, subtract 8 from the panel address to get the physical digital panel address.

Data verification method is 8-bit checksum.

Note: *Blank spaces in the above examples are added for visual clarity only. They do not actually appear in the command or in the response.*

2.6.2 Set I/O Configuration – Group Command G

Description:

Builds a table of module presence and type in isoLynx or Digital I/O Panel. Module present is 1, not present is 0 and location is encoded as bit position. Module types are Analog Input, Analog Output, Digital Input, Digital Output. If this command sets a channel to input or vacant, it will set that channel's average to zero. If this command sets a channel to output, it will also set that output to the current default output value from eeprom. However, it will not change the averaging sample weight or default output value in eeprom. Host software cross checks changes with existing I/O configuration and prompts user for verification of change. isoLynx firmware cross checks with the scan list table built using the Set Scan List command and sends error code for conflicts. This command writes I/O configuration to EEPROM.

Versions:

Analog and Digital

ASCII Command Format:

# characters	1	1	1	1	4	2 to 32	2	1
	>	i	P	G	MMMM	TT	DVF	CR

Where:

>	=	ASCII Character > (3E Hex)
i	=	isoLynx Address: 0 – F
P	=	Panel Address: 0 – F
		Analog Panel: 0 – 3
		Reserved: 4 – 7
		Digital Panel: 8 – F
G	=	Command Character (47 Hex)
MMMM	=	Modules to be configured
TT	=	Module Type
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

Channel Binary Format MMMM (4 Byte):

Data character	3	2	1	0
Channel #s	15 14 13 12	11 10 09 08	07 06 05 04	03 02 01 00
Example	0 0 0 0	1 0 1 0	0 0 0 0	0 1 0 1
Hex ASCII Char	0	A	0	5

Module type TT:

00 Hex	Analog or Digital Input
80 Hex	Analog or Digital Output
	(Analog or Digital is specified above in Panel Address)
	For each bit in MMMM that is set to one (1), there must be a corresponding data field TT.

ASCII Response Format:

Success Response Format

# characters	1	1	1	1	2	1
	A	i	P	G	DVF	CR

Where:

A	=	Good Response Character (41 Hex)(Ack)
i	=	isoLynx and Panel Address
P	=	isoLynx and Panel Address
G	=	Command Character (47 Hex)
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

Error Response Format

# characters	1	1	1	1	2	2	1
	N	i	P	G	EC	DVF	CR

Where:

N	=	Error Response Character (4E Hex)(Nack)
i	=	isoLynx and Panel Address
P	=	isoLynx and Panel Address
G	=	Command Character (47 Hex)
EC	=	Error Code (Reference Error Code Summary, Section 2.2.1.1)
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

Examples:

Command	> A1 G 0A05 80 80 00 00 1F CR
Response	A A1 G FA CR

Sends a Set I/O Configuration – Group command in ASCII format to the isoLynx Analog I/O Panel 1 at address A Hex. This example sets I/O configuration for channels 11, 09, 03, and 00 (see Channel Binary Format MMMM above). Channels 11 and 09 are selected to be outputs (80 Hex), and channels 03 and 00 are selected to be inputs (00 Hex).

Command	> A9 G 0A05 80 80 00 00 27 CR
Response	A A9 G 02 CR

Sends a Set I/O Configuration – Group command in ASCII format to the isoLynx Digital I/O Panel 1 at address A Hex. This example sets I/O configuration for channels 11, 09, 03, and 00 (see Channel Binary Format MMMM above). Channels 11 and 09 are selected to be outputs (80 Hex), and channels 03 and 00 are selected to be inputs (00 Hex).

NOTE: Since 8-F in the panel address indicates a digital panel, subtract 8 from the panel address to get the physical panel address.

Data verification method is 8-bit checksum.

Note: *Blank spaces in the above examples are added for visual clarity only. They do not actually appear in the command or in the response.*

2.7 Input Commands

2.7.1 Read Analog or Digital Default Output Values - Group Command *

Description:

This command reads the analog channels specified by MMMM or all digital channels set by the Set Analog or Digital Default Output Value command (&). The count (analog) or bit (digital) values read are returned in data field DDDD. For analog channels, the number of data fields DDDD returned will be equal to the number of 1's in MMMM and data field DDDD is a 16 bit signed integer (4 Hex characters (chars)). For digital channels, data for all 16 channels is returned in DDDD. The default values are retrieved from EEPROM. Factory default values are shown below.

Versions:

Analog and Digital

ASCII Command Format:

# characters	1	1	1	1	4	2	1
	>	i	P	*	MMMM	DVF	CR

Where:

>	=	ASCII Character > (3E Hex)
i	=	isoLynx Address: 0 – F
P	=	Panel Address: 0 – F
		Analog Panel: 0 – 3
		Reserved: 4 – 7
		Digital Panel: 8 – F
*	=	Command Character (2A Hex)
MMMM	=	Module(s) to be read
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

Channel Binary Format MMMM (4 Character):

Data character	3	2	1	0
Channel #s	15 14 13 12	11 10 09 08	07 06 05 04	03 02 01 00
Example	0 0 0 0	1 0 1 0	0 0 0 0	0 1 0 1
Hex ASCII Char	0	A	0	5

ASCII Response Format:

Success Response Format

# characters	1	1	1	1	4 to 64	2	1
	A	i	P	*	DDDD	DVF	CR

Where:

A = Good Response Character (41 Hex)(Ack)
i = isoLynx Address
P = Panel Address
* = Command Character (2A Hex)
DDDD = Analog:Hex Value of data in counts (4 Chars) for channels specified in MMMM Factory default value is 0 counts.
Digital: Data for all 16 channels (4 Chars)
Factory default value is logic 0
DVF = Data Verification Field
CR = Carriage Return (0D Hex)

Data Format (DDDD):

(Single Channel)

Data character	3	2	1	0
Example	0 0 0 0	0 0 0 0	0 0 0 0	1 0 1 0
Hex ASCII Char	0	0	0	A

Error Response Format

# characters	1	1	1	1	2	2	1
	N	i	P	*	EC	DVF	CR

Where:

N = Error Response Character (4E Hex)(Nack)
i = isoLynx Address
P = Panel Address
* = Command Character (2A Hex)
EC = Error Code (Reference Error Code Summary, Section 2.2.1.1)
DVF = Data Verification Field
CR = Carriage Return (0D Hex)

Examples:

Command > A1 * 0A05 00 D2 CR
Response A A1 * 0000 7FFF 8000 3CD0 58 CR

Sends a Read Default Output Values – Group command in ASCII format to the isoLynx Analog I/O Panel 1 at address A Hex. Current counts are requested from channels 11, 09, 03, and 00 (see Channel Binary Format MMMM above). Channel 11 returns 0V (0000 Hex), channel 09 returns +full scale +10V (7FFF Hex), channel 03 returns –full scale -10V (8000 Hex), and channel 00 returns +4.75V (3CD0 Hex).

Command > A9 * A4 CR
Response A A9 * 0204 AB CR

Sends a Read Default Output Values – Group command in ASCII format to the isoLynx Digital I/O Panel 1 at address A Hex. Channels 15 through 10 return a logic 0, channel 09 returns a logic 1, channels 08 through 04 return a logic 0, channel 02 returns a logic 1, and channels 01 and 00 return a logic 0. (For channel to bit state correlation, use Channel Binary Format table above.)

NOTE: Since 8-F in the panel address indicates a digital panel, subtract 8 from the panel address to get the physical digital panel address.

Data verification method is 8-bit checksum.

Note: *Blank spaces in the above examples are added for visual clarity only. They do not actually appear in the command or in the response.*

2.7.2 Read Analog or Digital Inputs - Group Command R

Description:

This command reads the analog channels specified by MMMM or all digital channels. For analog channels, the number of data fields DDDD in the response will be equal to the number of channels selected by entering 1's in the MMMM field and the data field will be 4 characters (chars) long per channel. Data type field TT can be specified as current counts or average counts. For digital channels, fields MMMM and TT are not used and data for all 16 channels is returned in DDDD. isoLynx firmware cross checks with I/O Configuration and sends error code for reading output channel or vacant channel.

Versions:

Analog and Digital

ASCII Command Format:

# characters	1	1	1	1	4	2	2	1
	>	i	P	R	MMMM	TT	DVF	CR

Where:

> = ASCII Character > (3E Hex)
> i = isoLynx Address: 0 – F
> P = Panel Address: 0 – F
> Analog Panel: 0 – 3
> Reserved: 4 – 7
> Digital Panel: 8 – F
> R = Command Character (52 Hex)
> MMMM = Module for which data is to be returned
> TT = Data type
> DVF = Data Verification Field
> CR = Carriage Return (0D Hex)

Channel Binary Format MMMM (4 Character):

Data character	3	2	1	0
Channel #s	15 14 13 12	11 10 09 08	07 06 05 04	03 02 01 00
Example	0 0 0 0	1 0 1 0	0 0 0 0	0 1 0 1
Hex ASCII Char	0	A	0	5

Data type TT:

00 Hex Current Counts (Analog Module)
01 Hex Average Counts (Analog Module)

ASCII Response Format:

Success Response Format

# characters	1	1	1	1	4 to 64	2	1
	A	i	P	R	DDDD	DVF	CR

Where:

A = Good Response Character (41 Hex)(Ack)
i = isoLynx Address
P = Panel Address
R = Command Character (52 Hex)
DDDD = Hex Value of channel counts (4 Characters) from the analog channels specified in MMMM or all digital channels.
DVF = Data Verification Field
CR = Carriage Return (0D Hex)

Data Format (DDDD):

(Single Channel)

Data character	3	2	1	0
Example	1 0 0 0	0 0 0 0	1 1 1 1	0 0 0 1
Hex ASCII Char	8	0	F	1

Error Response Format

# characters	1	1	1	1	2	2	1
	N	i	P	R	EC	DVF	CR

Where:

N = Error Response Character (4E Hex)(Nack)
i = isoLynx Address
P = Panel Address
R = Command Character (52 Hex)
EC = Error Code (Reference Error Code Summary, Section 2.2.1.1)
DVF = Data Verification Field
CR = Carriage Return (0D Hex)

Examples:

Command > A1 R 0A05 00 FA CR
Response A A1 R 0000 7FFF 8000 3CD0 80 CR

Sends a Read Inputs – Group command in ASCII format to the isoLynx Analog I/O Panel 1 at address A Hex. Current counts are requested from channels 11, 09, 03, and 00 (see Channel Binary Format MMMM above). Channel 11 returns 0V (0000 Hex), channel 09 returns +full scale +10V (7FFF Hex), channel 03 returns –full scale -10V (8000 Hex), and channel 00 returns +4.75V (3CD0 Hex).

Command > A9 R CC CR
Response A A9 R 0204 D3 CR

Sends a Read Inputs – Group command in ASCII format to the isoLynx Digital I/O Panel 1 at address A Hex. Channels 15 through 10 return a logic 0, channel 09 returns a logic 1, channels 08 through 04 return a logic 0, channel 02 returns a logic 1, and channels 01 and 00 return a logic 0. (For channel to bit state correlation, use Channel Binary Format table above.)

NOTE: Since 8-F in the panel address indicates a digital panel, subtract 8 from the panel address to get the physical digital panel address.

Data verification method is 8-bit checksum.

Note: *Blank spaces in the above examples are added for visual clarity only. They do not actually appear in the command or in the response.*

2.7.3 Read Analog or Digital Input Command r

Description:

This command reads the analog or digital channel specified in data field CC. For analog channels, the input value will be in data field DDDD in the response and the data field will be 4 characters (chars) long. Data type field TT can be specified as current counts or average counts. For digital channels, field TT is not used and the logic state of the channel is returned in D. isoLynx firmware cross checks with I/O Configuration and sends error code for reading an output channel or a vacant channel.

Versions:

Analog and Digital

ASCII Command Format:

# characters	1	1	1	1	2	2	2	1
	>	i	P	r	CC	TT	DVF	CR

Where:

> = ASCII Character > (3E Hex)
> i = isoLynx Address: 0 – F
> P = Panel Address: 0 – F
> Analog Panel: 0 – 3
> Reserved: 4 – 7
> Digital Panel: 8 – F
> r = Command Character (72 Hex)
> CC = Channel number to be read
> TT = Data type (see below)
> DVF = Data Verification Field
> CR = Carriage Return (0D Hex)

Data type TT:

00 Hex Current Counts (Analog Module)
01 Hex Average Counts (Analog Module)
(Analog or Digital is specified above in Panel Address)

ASCII Response Format:

Success Response Format

# characters	1	1	1	1	1 or 4	2	1
	A	i	P	r	DDDD	DVF	CR

Where:

A	=	Good Response Character (41 Hex)(Ack)
i	=	isoLynx Address
P	=	Panel Address
r	=	Command Character (72 Hex)
DDDD	=	Hex value of channel counts (4 characters) from the analog or logic value (1 character) of the digital channel specified by CC
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

Error Response Format

# characters	1	1	1	1	2	2	1
	N	i	P	r	EC	DVF	CR

Where:

N	=	Error Response Character (4E Hex)(Nack)
i	=	isoLynx Address
P	=	Panel Address
r	=	Command Character (72 Hex)
EC	=	Error Code (Reference Error Code Summary, Section 2.2.1.1)
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

Examples:

Command > A1 r 0B 00 B6 CR
Response A A1 r 3CD0 0F CR

Sends a Read Input command in ASCII format to the isoLynx Analog I/O Panel 1 at address A Hex. Current counts are requested from channel 11 (0B Hex). Channel 11 returns +4.75V (3CD0 Hex).

Command	> A9 r 0B 5E CR
Response	A A9 r 0 5D CR

Sends a Read Input command in ASCII format to the isoLynx Digital I/O Panel 1 at address A Hex. Channel 11 (0B Hex) returns a logic 0.

NOTE: Since 8-F in the panel address indicates a digital panel, subtract 8 from the panel address to get the physical digital panel address.

Data verification method is 8-bit checksum.

Note: *Blank spaces in the above examples are added for visual clarity only. They do not actually appear in the command or in the response.*

2.7.4 Read Averaging Sample Weight Command (

Description:

This command reads the averaging sample weight for the analog channel specified in data field CC. The input value will be in data field DDDD in the response and the data field will be 4 characters (chars) long. isoLynx firmware cross checks with I/O Configuration and sends error code for reading an output channel or a vacant channel.

Versions:

Analog

ASCII Command Format:

# characters	1	1	1	1	2	2	1
	>	i	P	(	CC	DVF	CR

Where:

>	=	ASCII Character > (3E Hex)
i	=	isoLynx Address: 0 – F
P	=	Panel Address: 0 – F
		Analog Panel: 0 – 3
		Reserved: 4 – 7
		Digital Panel: 8 – F
(	=	Command Character (28 Hex)
CC	=	Channel number to be read
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

ASCII Response Format:

Success Response Format

# characters	1	1	1	1	1 or 4	2	1
	A	i	P	(	DDDD	DVF	CR

Where:

A	=	Good Response Character (41 Hex)(Ack)
i	=	isoLynx Address
P	=	Panel Address
(	=	Command Character (28 Hex)
DDDD	=	Hex Value of averaging weight (4 Chars) for channel specified in CC. Averaging weights are only the discrete values: 0 & 2^0 through 2^14.
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

Error Response Format

# characters	1	1	1	1	2	2	1
	N	i	P	(	EC	DVF	CR

Where:

N	=	Error Response Character (4E Hex)(Nack)
i	=	isoLynx Address
P	=	Panel Address
(	=	Command Character (28 Hex)
EC	=	Error Code (Reference Error Code Summary, Section 2.2.1.1)
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

Examples:

Command	> A1 (0B 0C CR
Response	A A1 (4000 9F CR

Sends a Read Averaging Sample Weight command in ASCII format to the isoLynx Analog I/O Panel 1 at address A Hex. Averaging sample weight is requested from channel 11 (0B Hex). Channel 11 returns 4000 Hex (2^14).

Data verification method is 8-bit checksum.

Note: *Blank spaces in the above examples are added for visual clarity only. They do not actually appear in the command or in the response.*

2.8 Output Commands

2.8.1 Set Analog or Digital Default Output Values - Group Command &

Description:

This command sets the analog or digital output channels specified in data field MMMM to a default counts (analog) or bit (digital) value specified in data field DDDD. For analog channels, the number of data fields DDDD required will be equal to the number of 1's in MMMM. Data field DDDD is a 16 bit signed integer (4 Hex characters (chars)) for analog channels. The default values will be stored in EEPROM. Upon system reset due to watchdog timeout, brown-out, power cycling, or issue of the Reset command (B), these values will be read from EEPROM and used as default data for setting all output channels. Upon system reset due to a Reset Parameters to Default command ([ ), factory default values are read from ROM and also stored to EEPROM. Factory default values are shown below. isoLynx firmware (or host software) checks the I/O Configuration and sends an error code for writing an input channel or a vacant channel. Default values are read from EEPROM using the Read Default Output Values command (*).

Versions:

Analog and Digital

ASCII Command Format:

# characters	1	1	1	1	4	4 to 64	2	1
	>	i	P	&	MMMM	DDDD	DVF	CR

Where:

>	=	ASCII Character > (3E Hex)
i	=	isoLynx Address: 0 – F
P	=	Panel Address: 0 – F
		Analog Panel: 0 – 3
		Reserved: 4 – 7
		Digital Panel: 8 – F
&	=	Command Character (26 Hex)
MMMM	=	Module(s) to be set for Analog; not used for Digital
DDDD	=	Analog: Hex Value of data in counts (4 Chars) for channels specified in MMMM Factory default value is 0 counts. Digital: Data for all 16 channels (4 Chars) Factory default value is logic 0
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

Channel Binary Format MMMM (4 Character):

Data byte	3	2	1	0
Channel #s	15 14 13 12	11 10 09 08	07 06 05 04	03 02 01 00
Example	0 0 0 0	1 0 1 0	0 0 0 0	0 1 0 1
Hex ASCII Char	0	A	0	5

Data Format (DDDD):

(Single Channel)

Data character	3	2	1	0
Example	0 0 0 0	0 0 0 0	0 0 0 0	1 0 1 0
Hex ASCII Char	0	0	0	A

ASCII Response Format:

Success Response Format

# characters	1	1	1	1	2	1
	A	i	P	&	DVF	CR

Where:

A = Good Response Character (41 Hex)(Ack)
i = isoLynx Address
P = Panel Address
& = Command Character (26 Hex)
DVF = Data Verification Field
CR = Carriage Return (0D Hex)

Error Response Format

# characters	1	1	1	1	2	2	1
	N	i	P	&	EC	DVF	CR

Where:

N = Error Response Character (4E Hex)(Nack)
i = isoLynx Address
P = Panel Address
& = Command Character (26 Hex)
EC = Error Code (Reference Error Code Summary, Section 2.2.1.1)
DVF = Data Verification Field
CR = Carriage Return (0D Hex)

Examples:

Command > A1 & 0A05 0000 7FFF 8000 3CD0 E9 CR
Response A A1 & D9 CR

Sends a Set Default Output Values command in ASCII format to the isoLynx Analog I/O Panel 1 at address A Hex. Outputs are set on channels 11, 09, 03, and 00 (see Channel Binary Format MMMM above). Channel 11 sets to 0V (0000 Hex), channel 09 sets to +full scale +10V (7FFF Hex), channel 03 sets to –full scale -10V (8000 Hex), and channel 00 sets to +4.75V (3CD0 Hex).

Command > A9 & 0204 66 CR
Response A A9 & E1 CR

Sends a Set Default Output Values command in ASCII format to the isoLynx Digital I/O Panel 1 at address A Hex. Channels 15 through 10 set to a logic 0, channel 09 sets to a logic 1, channels 08 through 04 set to a logic 0, channel 02 sets to a logic 1, and channels 01 and 00 set to a logic 0. (For channel to bit state correlation, use Channel Binary Format table above.)

NOTE: Since 8-F in the panel address indicates a digital panel, subtract 8 from the panel address to get the physical panel address.

Data verification method is 8-bit checksum.

Note: *Blank spaces in the above examples are added for visual clarity only. They do not actually appear in the command or in the response.*

2.8.2 Set Analog or Digital Outputs - Group Command X

Description:

This command sets the analog or digital output channels specified in data field MMMM to the count (analog) or bit (digital) values specified in data field DDDD. For analog channels, the number of data fields DDDD required will be equal to the number of 1's in MMMM. Data field DDDD is a 16 bit signed integer (4 Hex characters (chars)) for analog channels. isoLynx firmware cross checks with I/O Configuration and sends error code for writing input channel or vacant channel.

Versions:

Analog and Digital

ASCII Command Format:

# characters	1	1	1	1	4	4 to 64	2	1
	>	i	P	X	MMMM	DDDD	DVF	CR

Where:

>	=	ASCII Character > (3E Hex)
i	=	isoLynx Address: 0 – F
P	=	Panel Address: 0 – F
		Analog Panel: 0 – 3
		Reserved: 4 – 7
		Digital Panel: 8 – F
X	=	Command Character (58 Hex)
MMMM	=	Module(s) to be set for Analog; not used for Digital
DDDD	=	Analog: Hex Value of data in counts (4 Bytes) for channels specified in MMMM Digital: Data for all 16 channels (4 Bytes)
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

Channel Binary Format MMMM (4 Character):

Data character	3	2	1	0
Channel #s	15 14 13 12	11 10 09 08	07 06 05 04	03 02 01 00
Example	0 0 0 0	1 0 1 0	0 0 0 0	0 1 0 1
Hex ASCII Char	0	A	0	5

Data Format (DDDD):

(Single Channel)

Data character	3	2	1	0
Example	0 0 0 0	0 0 0 0	0 0 0 0	1 0 1 0
Hex ASCII Char	0	0	0	A

ASCII Response Format:

Success Response Format

# characters	1	1	1	1	2	1
	A	i	P	X	DVF	CR

Where:

A = Good Response Character (41 Hex)(Ack)
i = isoLynx Address
P = Panel Address
X = Command Character (58 Hex)
DVF = Data Verification Field
CR = Carriage Return (0D Hex)

Error Response Format

# characters	1	1	1	1	2	2	1
	N	i	P	X	EC	DVF	CR

Where:

N = Error Response Character (4E Hex)(Nack)
i = isoLynx Address
P = Panel Address
X = Command Character (58 Hex)
EC = Error Code (Reference Error Code Summary, Section 2.2.1.1)
DVF = Data Verification Field
CR = Carriage Return (0D Hex)

Examples:

Command > A1 X 0A05 0000 7FFF 8000 3CD0 1B CR
Response A A1 X 0B CR

Sends a Set Output – Group command in ASCII format to the isoLynx Analog I/O Panel 1 at address A Hex. Outputs are set on channels 11, 09, 03, and 00 (see Channel Binary Format MMMM above). Channel 11 sets to 0V (0000 Hex), channel 09 sets to +full scale +10V (7FFF Hex), channel 03 sets to –full scale -10V (8000 Hex), and channel 00 sets to +4.75V (3CD0 Hex).

Command > A9 X 0204 98 CR
Response A A9 X 13 CR

Sends a Set Output – Group command in ASCII format to the isoLynx Digital I/O Panel 1 at address A Hex. Channels 15 through 10 set to a logic 0, channel 09 sets to a logic 1, channels 08 through 04 set to a logic 0, channel 02 sets to a logic 1, and channels 01 and 00 set to a logic 0. (For channel to bit state correlation, use Channel Binary Format table above.)

NOTE: Since 8-F in the panel address indicates a digital panel, subtract 8 from the panel address to get the physical panel address.

Data verification method is 8-bit checksum.

Note: *Blank spaces in the above examples are added for visual clarity only. They do not actually appear in the command or in the response.*

2.8.3 Set Analog or Digital Output Command x

Description:

This command sets the analog output value specified in counts for a single analog output channel or group of digital output channels specified.

This command sets the analog or digital output channel specified in data field CC to the count (analog) or bit (digital) values specified in data field DDDD. Data field DDDD is a 16 bit signed integer (4 Hex characters (chars)) for analog channels. isoLynx firmware cross checks with I/O Configuration and sends error code for writing input channel or vacant channel.

Versions:

Analog and Digital

ASCII Command Format:

# characters	1	1	1	1	2	1 or 4	2	1
	>	i	P	x	CC	DDDD	DVF	CR

Where:

>	=	ASCII Character > (3E Hex)
i	=	isoLynx Address: 0 – F
P	=	Panel Address: 0 – F Analog Panel: 0 – 3 Reserved: 4 – 7 Digital Panel: 8 – F
x	=	Command Character (78 Hex)
CC	=	Channel number to be set
DDDD	=	Analog: Hex Value of data in counts (4 Chars) for channel specified Digital: Data for channel specified (1 Char)
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

Data Format (DDDD):

(Single Channel)

Data character	3	2	1	0
Example	0 0 0 0	0 0 0 0	0 0 0 0	1 0 1 0
Hex ASCII Char	0	0	0	A

ASCII Response Format:

Success Response Format

# characters	1	1	1	1	2	1
	A	i	P	x	DVF	CR

Where:

A	=	Good Response Character (41 Hex)(Ack)
i	=	isoLynx Address
P	=	Panel Address
x	=	Command Character (78 Hex)
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

Error Response Format

# characters	1	1	1	1	2	2	1
	N	i	P	x	EC	DVF	CR

Where:

N	=	Error Response Character (4E Hex)(Nack)
i	=	isoLynx Address
P	=	Panel Address
x	=	Command Character (78 Hex)
EC	=	Error Code (Reference Error Code Summary, Section 2.2.1.1)
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

Examples:

Command > A1 x 0A 3CD0 45 CR
Response A A1 x 2B CR

Sends a Set Output command in ASCII format to the isoLynx Analog I/O Panel 1 at address A Hex. The output is set on channel 10. Channel 10 sets to +4.75V.

Command	> A9 x 0A 1 94 CR
Response	A A9 x 33 CR

Sends a Set Output command in ASCII format to the isoLynx Digital I/O Panel 1 at address A Hex. Channel 10 sets to a logic 1.

NOTE: Since 8-F in the panel address indicates a digital panel, subtract 8 from the panel address to get the physical panel address.

Data verification method is 8-bit checksum.

Note: *Blank spaces in the above examples are added for visual clarity only. They do not actually appear in the command or in the response.*

2.8.4 Set Averaging Sample Weight Command h

Description:

This command sets the averaging sample weight specified by data field DDDD for analog input channel specified in data field CC. Data field DDDD is a 16 bit unsigned integer (4 Hex characters (chars)). This command writes any changed sample weight value to EEPROM. Each channel specified is checked by isoLynx firmware against I/O configuration table and only analog input channels are set. Attempting to set a Digital I/O channel will result in an error response.

Versions:

Analog

ASCII Command Format:

# characters	1	1	1	1	2	4	2	1
	>	i	P	h	CC	DDDD	DVF	CR

Where:

>	=	ASCII Character > (3E Hex)
i	=	isoLynx Address: 0 – F
P	=	Panel Address: 0 – F Analog Panel: 0 – 3 Reserved: 4 – 7 Digital Panel: 8 – F
h	=	Command Character (68 Hex)
CC	=	Channel Number (00 to 0F Hex)
DDDD	=	Hex Value of averaging weight (4 Chars) for channel specified in CC. Averaging weights are only the discrete values: 0 & 2^0 through 2^14. Weight value = 0 discontinues averaging causing last calculated average to remain in RAM. Weight value = 1 causes average values to track current values.
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

DATA Format (DDDD):

(Single Channel)

Data character	3	2	1	0
Example	0 0 0 0	0 0 0 0	0 0 0 0	1 0 1 0
Hex ASCII Char	0	0	0	A

ASCII Response Format:

Success Response Format

# characters	1	1	1	1	2	1
	A	i	P	h	DVF	CR

Where:

A	=	Good Response Character (41 Hex)(Ack)
i	=	isoLynx Address
P	=	Panel Address
h	=	Command Character (68 Hex)
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

Error Response Format

# characters	1	1	1	1	2	2	1
	N	i	P	h	EC	DVF	CR

Where:

N	=	Error Response Character (4E Hex)(Nack)
i	=	isoLynx Address
P	=	Panel Address
h	=	Command Character (68 Hex)
EC	=	Error Code (Reference Error Code Summary, Section 2.2.1.1)
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

Examples:

Command > A1 h 0A 0020 0D CR
Response A A1 h 1B CR

Sends a Set Averaging Sample Weight command in ASCII format to the isoLynx Analog I/O Panel 1 at address A Hex. The example command sets a weight of 0020 Hex or 2^5 = 32 Decimal for channel 10 (0A Hex).

Data verification method is 8-bit checksum.

Note: *Blank spaces in the above examples are added for visual clarity only. They do not actually appear in the command or in the response.*

2.9 Network Commands

2.9.1 Read Ethernet Configuration Command +

Description:

Reads from EEPROM the bits in the Ethernet Configuration Control Bytes which configure the Ethernet port.

Versions:

Analog

ASCII Command Format:

# characters	1	1	1	1	2	1
	>	i	P	+	DVF	CR

Where:

>	=	ASCII Character > (3E Hex)
i	=	isoLynx Address is: 0 – F
P	=	Panel Address is: 0 – F
		Analog Panel: 0 – 3
		Reserved: 4 – 7
		Digital Panel: 8 – F
+	=	Command Character (2B Hex)
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

ASCII Response Format:

Success Response Format

# characters	1	1	1	1	8	8	8	8	8	24	2	1
	>	i	P	+	IP	sn	gw	dns	dhcp	res	DVF	CR

Where:

A	=	Good Response Character (41 Hex)(Ack)
i	=	isoLynx Address
P	=	Panel Address
+	=	Command Character (2B Hex)
IP	=	Internet Protocol address (4 bytes)
sn	=	subnet mask (4 bytes)
gw	=	gateway address (4 bytes)
dns	=	domain name server (4 bytes)
dhcp	=	dynamic host configuration protocol (4 bytes)
res	=	reserved (12 bytes)
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

Error Response Format

# characters	1	1	1	1	2	2	1
	N	i	P	+	EC	DVF	CR

Where:

N	=	Error Response Character (4E Hex)(Nack)
i	=	isoLynx Address
P	=	Panel Address
+	=	Command Character (2B Hex)
EC	=	Error Code (Reference Error Code Summary, Section 2.2.1.1)
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

Examples:

Command	>	A0 + 9C CR
Response	A	A0 + C0 A8 FE 1E FF FF FF 00 C0 A8 FE FE D8 E7 29 16 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0E9A CR

Sends a Read Ethernet Configuration command in ASCII format to the isoLynx Analog I/O Panel 0 at address A Hex. Four each values are returned for IP address, subnetmask, gateway, and dnsserver, respectively.

Data verification method is 8-bit checksum.

Note: *Blank spaces in the above examples are added for visual clarity only. They do not actually appear in the command or in the response.*

2.9.2 Set Ethernet Configuration Command

Description:

Set or clear the bits in the Ethernet Configuration Control Bytes which configure the Ethernet port. This command writes changed Ethernet Configuration values to EEPROM.

Versions:

Analog

ASCII Command Format:#

characters	1	1	1	1	8	8	8	8	8	24	2	1
	>	i	P	#	IP	sn	gw	dns	dhcp	res	DVF	CR

Where:

>	=	ASCII Character > (3E Hex)
i	=	isoLynx Address: 0 – F
P	=	Panel Address: 0 – F
		Analog Panel: 0 – 3
		Reserved: 4 – 7
		Digital Panel: 8 – F
#	=	Command Character (23 Hex)
IP	=	Internet Protocol address (4 bytes)
sn	=	subnet mask (4 bytes)
gw	=	gateway address (4 bytes)
dns	=	domain main server (4 bytes)
dhcp	=	dynamic host control protocol (4 bytes)
res	=	reserved (12 bytes)
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

ASCII Response Format:

Success Response Format

# characters	1	1	1	1	2	1
	A	i	P	#	DVF	CR

Where:

A	=	Good Response Character (41 Hex)(Ack)
i	=	isoLynx Address
P	=	Panel Address
#	=	Command Character (23 Hex)
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

Error Response Format

# characters	1	1	1	1	2	2	1
	N	i	P	#	EC	DVF	CR

Where:

N	=	Error Response Character (4E Hex)(Nack)
i	=	isoLynx Address
P	=	Panel Address
#	=	Command Character (23 Hex)
EC	=	Error Code (Reference Error Code Summary, Section 2.2.1.1)
DVF	=	Data Verification Field
CR	=	Carriage Return (0D Hex)

Examples:

Command	>	A0 # C0 A8 FE 1E FF FF FF 00 C0 A8 FE FE D8 E7 29 16 00 00 00 00 00 00 00 00 00 00 00 00 51 CR
Response	A	A0 # D5 CR

Sends a Set Ethernet Configuration command in ASCII format to the isoLynx Analog I/O Panel 0 at address A Hex. Four each values are set for IP address, subnetmask, gateway, and dnsserver, respectively.

Data verification method is 8-bit checksum.

Note: *Blank spaces in the above examples are added for visual clarity only. They do not actually appear in the command or in the response.*

2.10 Usage Examples

2.10.1 Configuring and Running Input and Output Channels

You have an isoLynx completely connected, powered, initialized, and several modules inserted. You are considering using a programming language such as Visual C++ or Visual BASIC to create software to sequence through a set of commands which acquire data and possibly even set some outputs to control a process. The following sequence of commands illustrates one possible set of commands you might use in a simple application.

* Send a Reset (B) or Reset Parameters to Default ([ ) command. The Reset command restarts the isoLynx operating system and uses parameters, such as default outputs and average weights that have been previously stored in EEPROM. The Reset to Default command restarts the isoLynx operating system and uses parameters, such as default outputs and average weights that are the Factory Defaults stored in ROM. Reference the reset commands in Section 2.5 for more details. This step is optional.

* Send a Set I/O Configuration – Group (G) command or series of commands depending on the number of channels you need to configure and whether or not you have Digital I/O channels in your system. Optionally, you may wish to send Read I/O Configuration (Y) commands first, then send the Set I/O Configuration commands to configure new channels or reconfigure existing channels. Reference the I/O Configuration commands in Section 2.6 for more details.

* If you need to initialize or change default outputs, you may at this point wish to send a Set Default Output Values – Group (&) command or series of commands depending on the number of channels that need default outputs and whether or not you have Digital I/O channels in your system. Optionally, you may wish to send Read Default Output Values (*) commands first, then send the Set Default Output Values commands to initialize or change default outputs. Reference the Read and Set Default Output Values commands in Sections 2.7 and 2.8, respectively, for more details.

* At this point, you are ready to implement the data acquisition and/or control loop part of your program. Here you will be using Read Inputs (R or r) commands to acquire data, analog and/or digital, and analog averages. If you are implementing a control application, you will also have your software checking input data and/or averages and conditionally sending Set Output (X or x) commands, analog and/or digital. Reference the Read Inputs and Set Outputs commands in Sections 2.7 and 2.8, respectively, for more details.

2.10.2 Changing from RS-232 to Ethernet and back

You have an isoLynx completely connected, powered, initialized, and several modules configured. You have been running some data samples using RS-232 and now want to deploy the system in an Ethernet network. If you have not installed an Ethernet Board in your isoLynx, refer to the isoLynx™ Hardware User Manual section 4.1.2 and following subsections for installation instructions. Then return here to complete the software transformation. Here is one possible sequence of actions you might use to accomplish the change using the isoLynx Command Protocol:

* Send a Reset (B) or Reset Parameters to Default ([ ) command depending on whether you want to use the last parameters the isoLynx stored from your system checkout or you want to use some known default parameters the isoLynx stored in response to some set default commands you used in your parameter initialization. Reference the reset commands in Section 2.5. This step is optional.

* Send a Set Ethernet Configuration (#) command to set up the correct IP addresses, subnet masks, etc. for your particular network configuration. You may also want to send a Read Ethernet Configuration (+) command at this point to verify the Ethernet parameters. Reference the Ethernet configuration commands in Section 2.9.

* Send a Set System Parameters (@) command with the NI (Network Interface) parameter set to Ethernet (3). Reference the Set System Parameters command in Section 2.5.

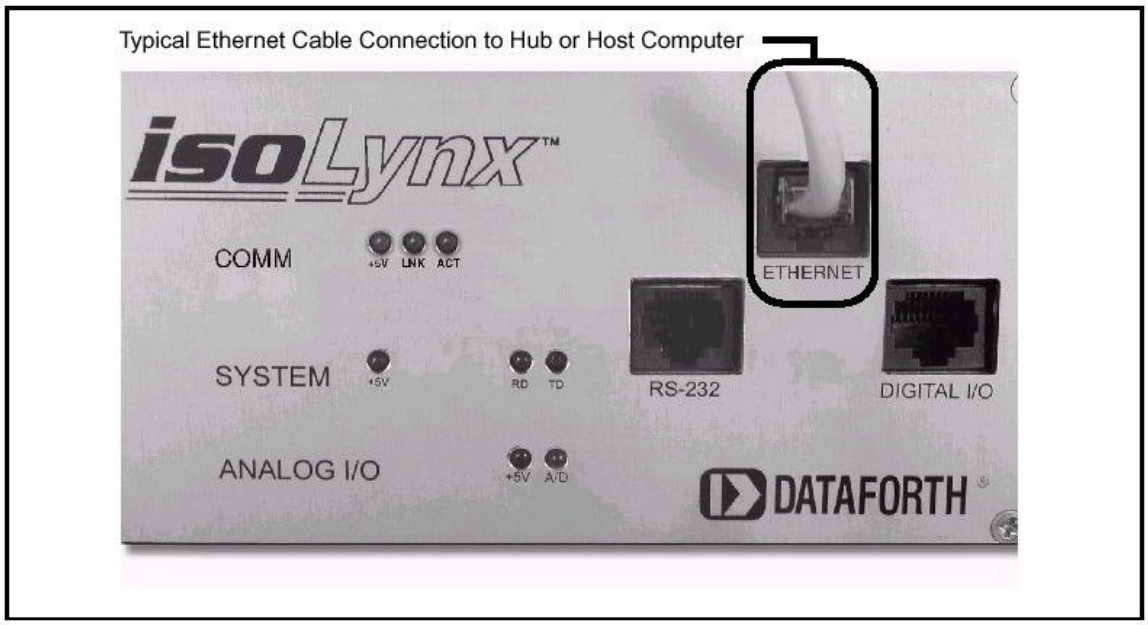


Figure 2.10.2-1 Typical Ethernet Cable Connection

* Plug a Cat 5 patch cable or isoLynx SLX141 cable of the appropriate length into the Ethernet port (RJ-45) on the isoLynx Analog I/O Base Unit System Enclosure. Plug the other end of the cable (RJ-45) into an open port on your Ethernet hub. If your network connection is directly to an Ethernet port in the host computer, use a Cat 5 crossover patch cable or an isoLynx SLX141-X cable. The isoLynx is now ready to exchange data over the Ethernet network.

Later, you may want to move an isoLynx from the Ethernet environment to an RS-232 link. Here is one possible sequence of commands you might use to accomplish the change using the isoLynx Command Protocol:

* Send a Reset (B) or Reset Parameters to Default ([ ) command depending on whether you want to use the last parameters the isoLynx stored from your Ethernet session or you want to use some known default parameters the isoLynx stored in response to some set default commands you used in your parameter initialization. Reference the reset commands in Section 2.5. This step is optional.

* Send a Set System Parameters (@) command with the NI (Network Interface) parameter set to RS-232 (0), CF (Comm. Config.) set to RS-232 (0), and DR (Data Rate) set to the data rate you need. Reference the Set System Parameters command in Section 2.5.

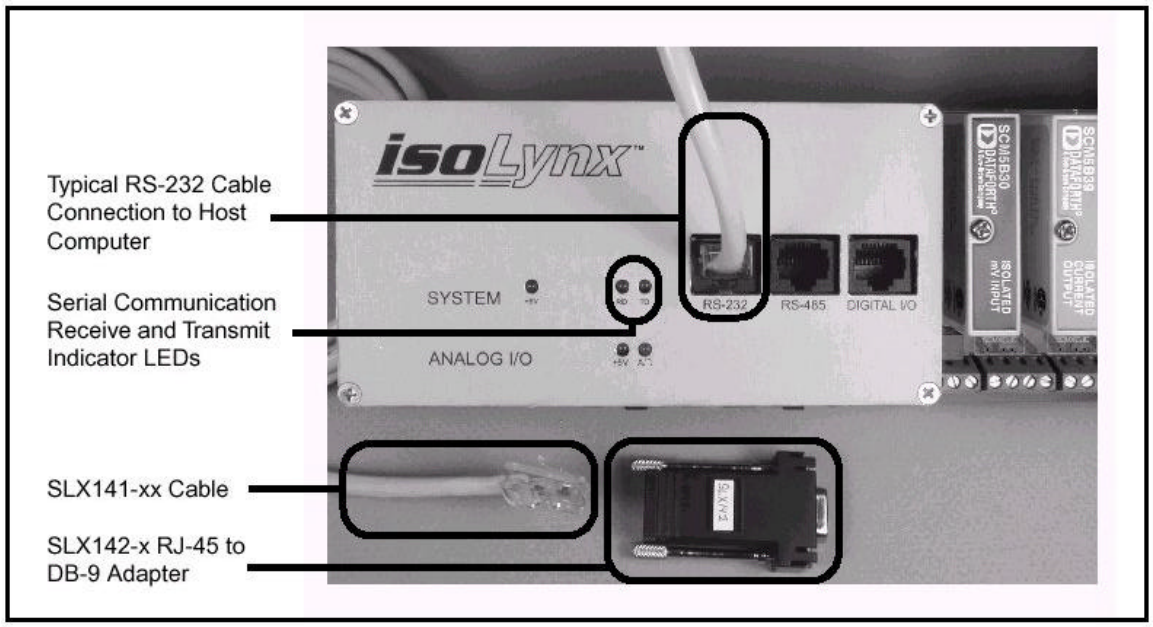


Figure 2.10.2-2 Typical RS-232 Cable Connection

* Disconnect the Ethernet cable from the port (RJ-45) on the Ethernet hub and from the Ethernet port (RJ-45) on the isoLynx Analog I/O Base Unit System Enclosure. Plug one end of that cable into the RS-232 port (RJ-45) on the isoLynx Analog I/O Base Unit System Enclosure. Plug the other end into the RJ-45 jack in an RJ-45 to DB-9 RS-232 adapter SLX142. Plug the adapter into an RS-232 port on your host computer. The isoLynx is now ready to communicate over RS-232.

2.10.3 Changing from RS-232 to RS-485 2-wire and back

You have an isoLynx completely connected, powered, initialized, and several modules configured. You have been running some data samples using RS-232 and now want to deploy the system in an RS-485 2-wire network. Here is one possible sequence of actions you might use to accomplish the change using the isoLynx Command Protocol:

* Send a Reset (B) or Reset Parameters to Default ([ ) command depending on whether you want to use the last parameters the isoLynx stored from your system checkout or you want to use some known default parameters the isoLynx stored in response to some set default commands you used in your parameter initialization. Reference the reset commands in Section 2.5. This step is optional.

* Send a Set System Parameters (@) command with the NI (Network Interface) parameter set to RS-485 2-wire (1), CF (Comm. Config.) set to the RS-485 network configuration you need, and DR (Data Rate) set to the data rate you need. Reference the Set System Parameters command in Section 2.5.

* Set your host computer's RS-485 adapter card to RS-485 2-wire operation.

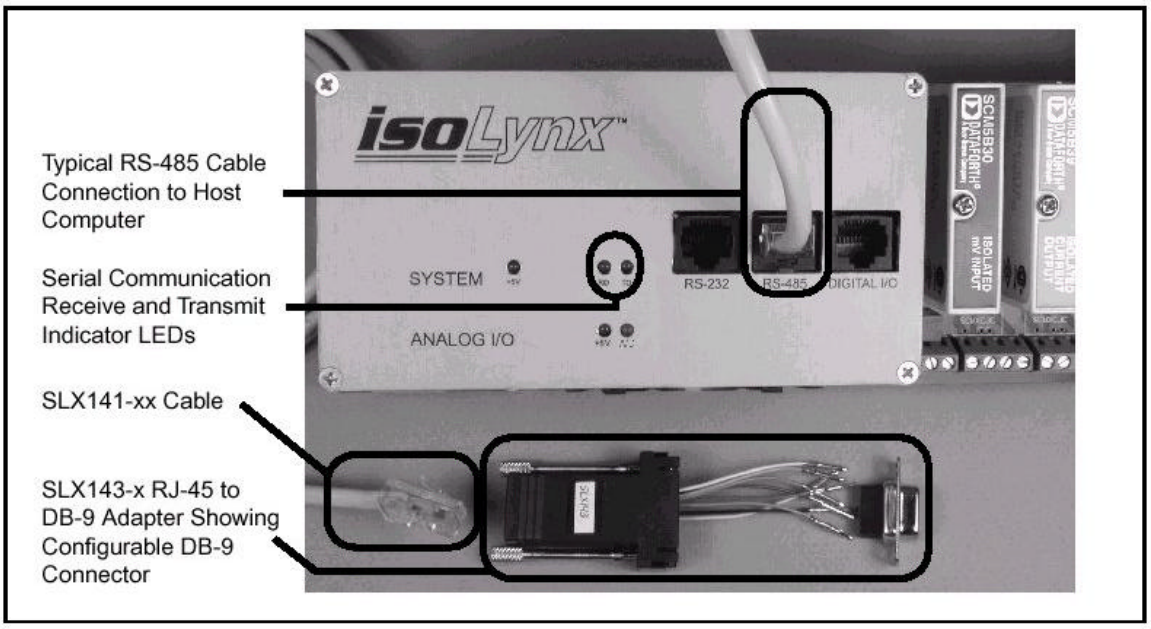


Figure 2.10.3-1 Typical RS-485 Cable Connection

* Plug a Cat 5 patch cable or isoLynx SLX141 cable of the appropriate length into the RS-485 port (RJ-45) on the isoLynx Analog I/O Base Unit System Enclosure. Plug the other end into the RJ-45 jack in an uncommitted RJ-45 to DB-9 adapter SLX143 which you have previously wired. Reference the isoLynx™ Hardware User Manual for pinout information. Plug the adapter into a port on your host computer's RS-485 adapter card. Remember to switch in/out the RS-485 terminations on the isoLynx Processor Board and on your host computer's RS-485 adapter card as needed. Reference the isoLynx™ Hardware User Manual. The isoLynx is now ready to communicate over RS-485 2-wire.

Later, you may want to move an isoLynx from the RS-485 2-wire environment to an RS-232 link. Here is one possible sequence of commands you might use to accomplish the change using the isoLynx Command Protocol:

* Send a Reset (B) or Reset Parameters to Default ([ ) command depending on whether you want to use the last parameters the isoLynx stored from your Ethernet session or you want to use some known default parameters the isoLynx stored in response to some set default commands you used in your parameter initialization. Reference the reset commands in Section 2.5. This step is optional.

* Send a Set System Parameters (@) command with the NI (Network Interface) parameter set to RS-232 (0) , CF (Comm. Config.) set to RS-232 (0), and DR (Data Rate) set to the data rate you need. Reference the Set System Parameters command in Section 2.5.

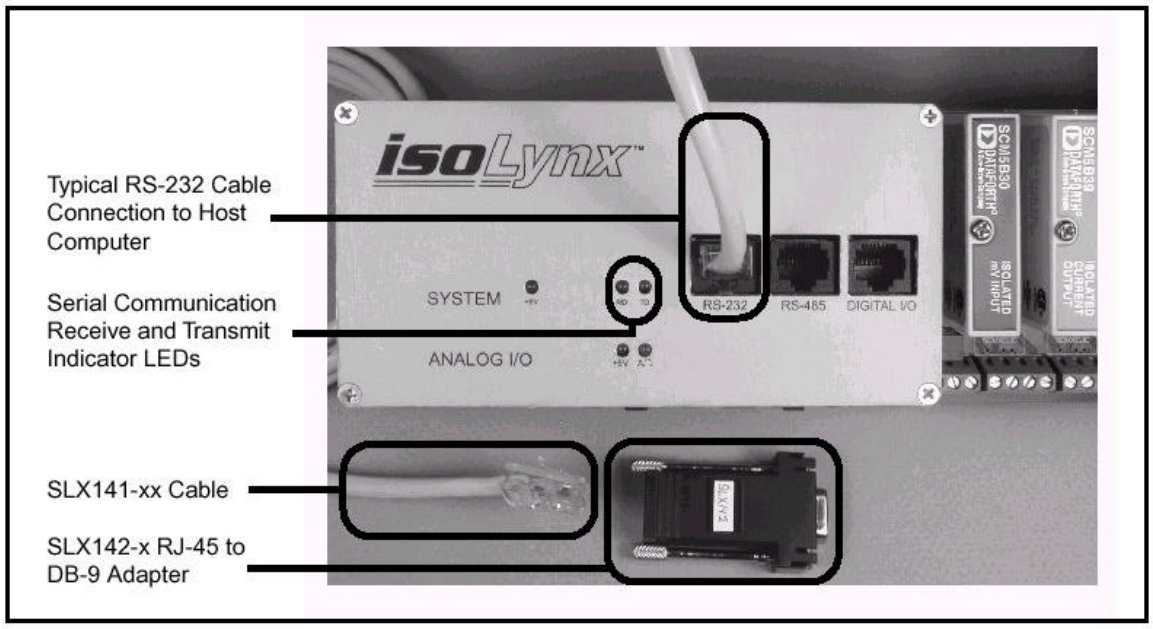


Figure 2.10.3-2 Typical RS-232 Cable Connection

* Disconnect the RS-485 2-wire cable and adapter from the port (DB-9) on the host computer. Disconnect the cable from the RJ-45 jack in your RS-485 2-wire RJ-45 to DB-9 adapter and plug the cable into the RS-232 port (RJ-45) on the isoLynx Analog I/O Base Unit System Enclosure. Plug the other end into the RJ-45 jack in an RJ-45 to DB-9 RS-232 adapter SLX142. Plug the adapter into an RS-232 port on your host computer. The isoLynx is now ready to communicate over RS-232.

2.10.4 Changing from RS-485 2-wire to Ethernet and back

You have an isoLynx completely connected, powered, initialized, and several modules configured. You have been running some data samples using RS-485 2-wire and now want to deploy the system in an Ethernet network. Here is one possible sequence of actions you might use to accomplish the change. Reference the isoLynx™ Hardware User Manual section 4.1.2 “Opening the isoLynx Controller, etc.” and following sections to subsection “Field Installation Instructions” for the hardware modifications. Then return here to complete the transformation, using the isoLynx Command Protocol:

* Send a Reset (B) or Reset Parameters to Default ([ ) command depending on whether you want to use the last parameters the isoLynx stored from your system checkout or you want to use some known default parameters the isoLynx stored in response to some set default commands you used in your parameter initialization. Reference the reset commands in Section 2.5. This step is optional.

* Send a Set Ethernet Configuration (#) command to set up the correct IP addresses, subnet masks, etc. for your particular network configuration. You may also want to send a Read Ethernet Configuration (+) command at this point to verify the Ethernet parameters. Reference the Ethernet configuration commands in Section 2.9.

* Send a Set System Parameters (@) command with the NI (Network Interface) parameter set to Ethernet (3). Reference the Set System Parameters command in Section 2.5.

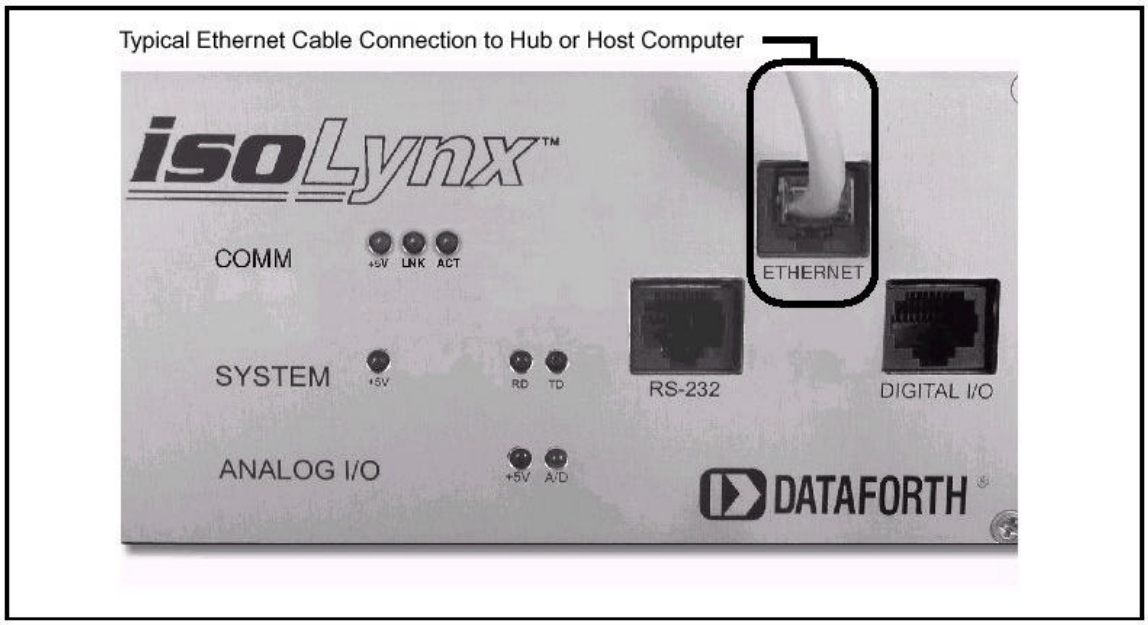


Figure 2.10.4-1 Typical Ethernet Cable Connection

* Plug a Cat 5 patch cable or isoLynx SLX141 cable of the appropriate length into the Ethernet port (RJ-45) on the isoLynx Analog I/O Base Unit System Enclosure. Plug the other end of the cable (RJ-45) into an open port on your Ethernet hub. If your network connection is directly to an Ethernet port in the host computer, use a Cat 5 crossover patch cable or an isoLynx SLX141-X cable. The isoLynx is now ready to exchange data over the Ethernet network.

Later, you may want to move an isoLynx from the Ethernet environment to an RS-485 2-wire link. Here is one possible sequence of commands you might use to accomplish the change. Reference the isoLynx™ Hardware User Manual section 4.1.2 “Opening the isoLynx Controller, etc.” and following sections to subsection “Field Installation Instructions”. Skip to the paragraph in the subsection starting “If for any reason,” for the hardware modifications. Then return here to complete the transformation, using the isoLynx Command Protocol:

* Send a Reset (B) or Reset Parameters to Default ([ ) command depending on whether you want to use the last parameters the isoLynx stored from your Ethernet session or you want to use some known default parameters the isoLynx stored in response to some set default commands you used in your parameter initialization. Reference the reset commands in Section 2.5. This step is optional.

* Send a Set System Parameters (@) command with the NI (Network Interface) parameter set to RS-485 2-wire (1), CF (Comm. Config.) set to the RS-485

network configuration you need, and DR (Data Rate) set to the data rate you need. Reference the Set System Parameters command in Section 2.5.

* Set your host computer's RS-485 adapter card to RS-485 2-wire operation.

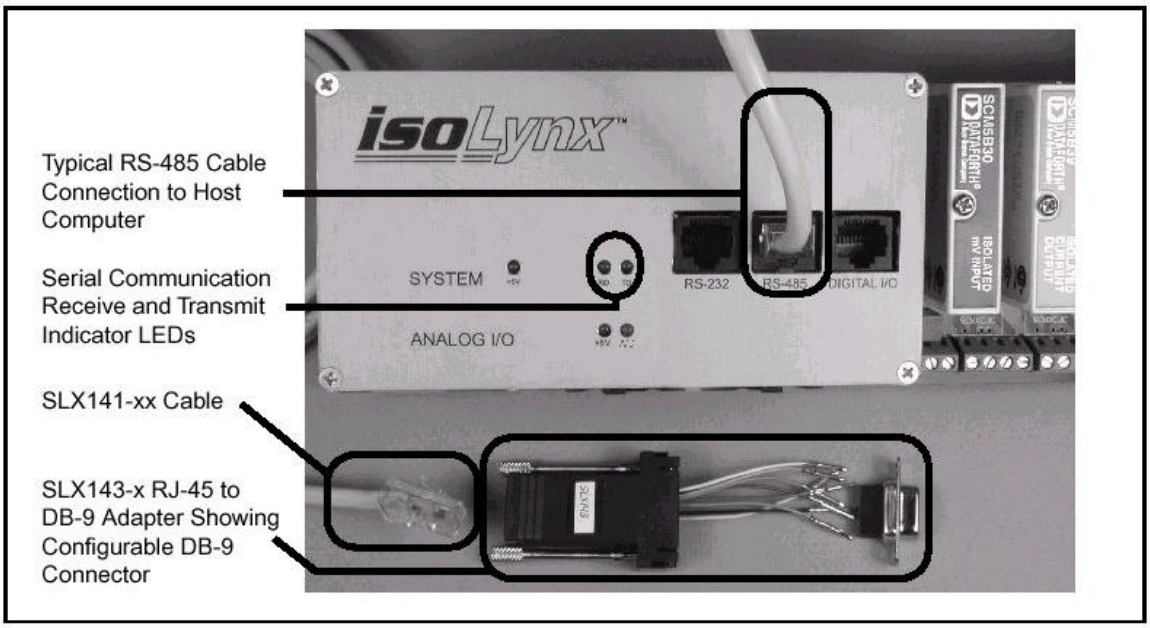


Figure 2.10.4-2 Typical RS-485 Cable Connection

* Disconnect the Ethernet cable from the port (RJ-45) on the Ethernet hub and from the Ethernet port (RJ-45) on the isoLynx Analog I/O Base Unit System Enclosure. Plug one end of that cable into the RS-485 port (RJ-45) on the isoLynx Analog I/O Base Unit System Enclosure. Plug the other end into the RJ-45 jack in an uncommitted RJ-45 to DB-9 adapter SLX143 which you have previously wired. Reference the isoLynx^{TM} Hardware User Manual for pinout information. Plug the adapter into a port on your host computer's RS-485 adapter card. Remember to switch in/out the RS-485 terminations on the isoLynx Processor Board and on your host computer's RS-485 adapter card as needed. Reference the isoLynx^{TM} Hardware User Manual. The isoLynx is now ready to communicate over RS-485 2-wire.

2.10.5 Changing from RS-485 2-wire to 4-wire and back

You have an isoLynx completely connected, powered, initialized, and several modules configured. You have been running some data samples using RS-485 2-wire and now want to deploy the system in an RS-485 4-wire network. Here is one possible sequence of actions you might use to accomplish the change using the isoLynx Command Protocol:

* Send a Reset(B) or Reset Parameters to Default([ ) command depending on whether you want to use the last parameters the isoLynx stored from your system checkout or you want to use some known default parameters the isoLynx stored in response to some set default commands you used in your parameter initialization. Reference the reset commands in Section 2.5. This step is optional.

* Send a Set System Parameters (@) command with the NI (Network Interface) parameter set to RS-485 4-wire (2). Reference the Set System Parameters command in Section 2.5.

* Set your host computer's RS-485 adapter card to RS-485 4-wire operation.

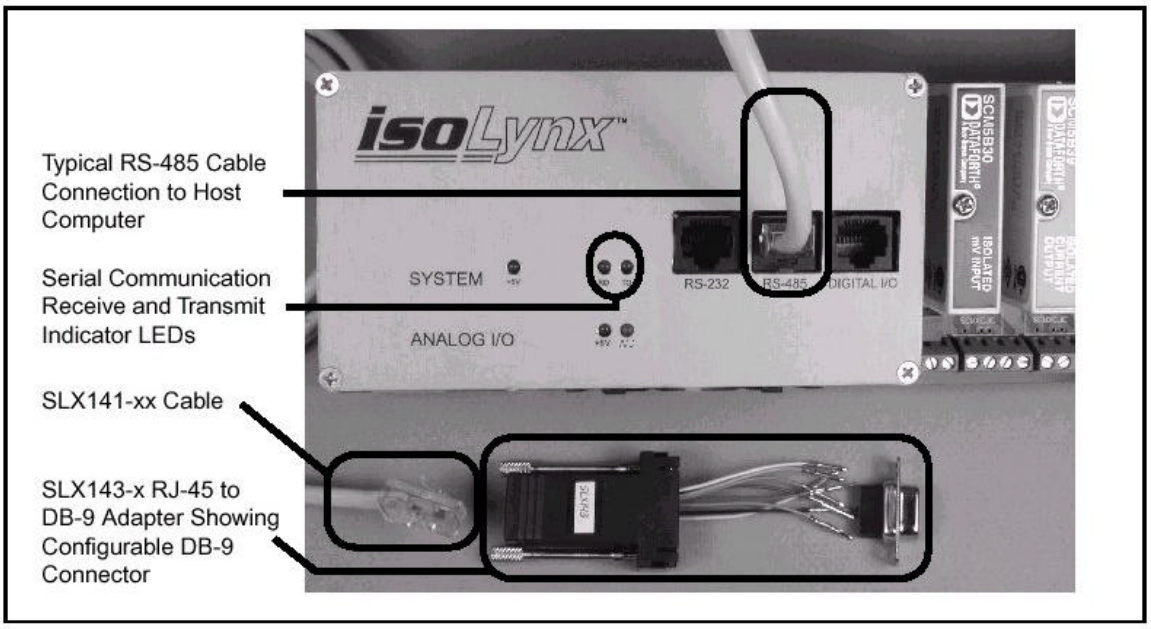


Figure 2.10.5-1 Typical RS-485 Cable Connection

* Plug a Cat 5 patch cable or isoLynx SLX141 cable of the appropriate length into the RS-485 port (RJ-45) on the isoLynx Analog I/O Base Unit System Enclosure. Plug the other end into the RJ-45 jack in an uncommitted RJ-45 to DB-9 adapter SLX143 which you have previously wired. Reference the isoLynx™ Hardware User Manual for pinout information. Plug the adapter into a port on your host computer's RS-485 adapter card. Remember to switch in/out the RS-485 terminations on the isoLynx Processor Board and on your host computer's RS-485 adapter card as needed. Reference the isoLynx™ Hardware User Manual. The isoLynx is now ready to communicate over RS-485 4-wire.

Later, you may want to move an isoLynx from the RS-485 4-wire environment to an RS-485 2-wire link. Here is one possible sequence of commands you might use to accomplish the change using the isoLynx Command Protocol:

* Send a Reset (B) or Reset Parameters to Default ([ ) command depending on whether you want to use the last parameters the isoLynx stored from your RS-485 4-wire session or you want to use some known default parameters the isoLynx stored in response to some set default commands you used in your parameter initialization. Reference the reset commands in Section 2.5. This step is optional.

* Send a Set System Parameters (@) command with the NI (Network Interface) parameter set to RS-485 2-wire (1), CF (Comm. Config.) set to the RS-485 network configuration you need, and DR (Data Rate) set to the data rate you need. Reference the Set System Parameters command in Section 2.5.

* Set your host computer's RS-485 adapter card to RS-485 2-wire operation.

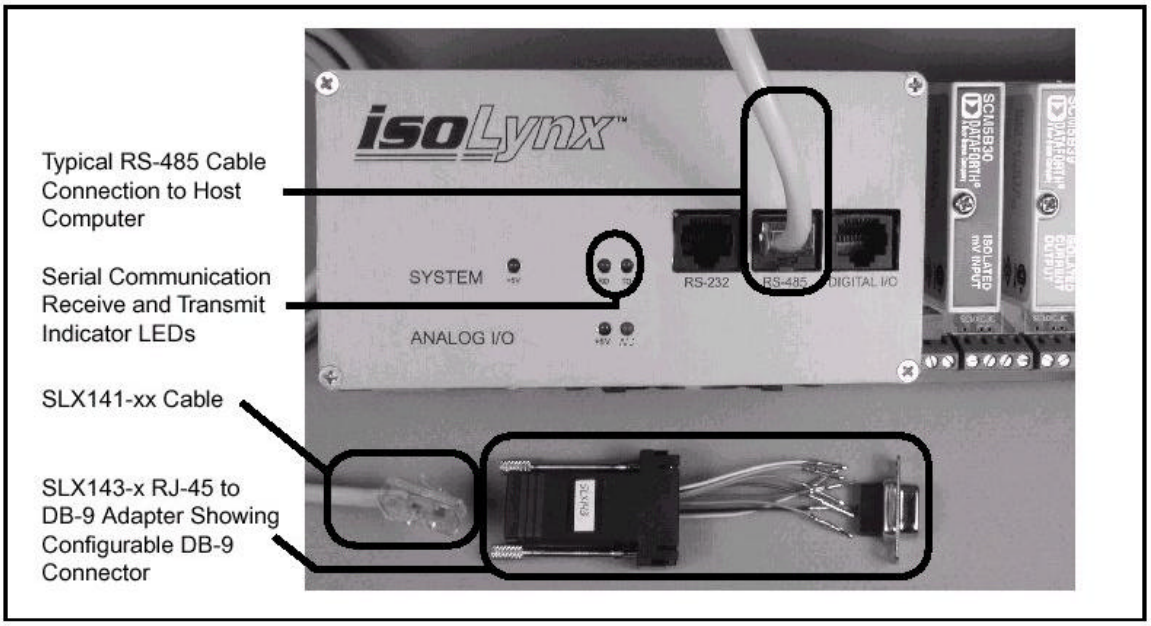


Figure 2.10.5-2 Typical RS-485 Cable Connection

* Disconnect the RS-485 4-wire cable and adapter from the port (DB-9) on the host computer. Disconnect the cable from the RJ-45 jack in your RS-485 4-wire RJ-45 to DB-9 adapter and plug it into your RS-485 2-wire RJ-45 to DB-9 adapter. Reference the isoLynx™ Hardware User Manual for pinout information. Plug the adapter into a port on your host computer's RS-485 adapter card. Remember to switch in/out the RS-485 terminations on the isoLynx Processor Board and on your host computer's RS-485 adapter card as needed. Reference the isoLynx™ Hardware User Manual. The isoLynx is now ready to communicate over RS-485 2-wire.

3 isoLynx Data Acquisition Library

The isoLynx Data Acquisition Library is a comprehensive library of functions with an associated Application Programming Interface (API). It can be used to develop custom 32-bit data acquisition and/or control applications for the Windows operating system. The library encapsulates the low-level communication details of the isoLynx Command Protocol (see Section 2, isoLynx Command Reference) thereby making it unnecessary for a programmer to have knowledge of such details.

3.1 Library Organization

The isoLynx library is organized into several distinct modules. A brief description of this organizational structure is given by this section. All files described here may be found on the installation CD in the CD:\LIB directory.

- LYNXW32.DLL is the main interface module. Files associated with this module are LYNXW32.LIB (import library), LYNXLIB.H (C++ include file), LYNXLIB.BAS (Visual Basic module), LYNXLIB.INI (default configuration data), and LYNXLIB.ERR (error file). Applications are statically linked to this module, and this module dynamically links to other library modules. See Section 3.2 and 3.3 for detailed linking instructions.

- _ISOLYNX.DLL encapsulates isoLynx specific functionality. Files associated with this module are ISOLYNX.H (C++ include file), ISOLYNX.BAS (Visual Basic module), ISOLYNX.INI (default configuration data) and ISOLYNX.ERR (error file).

- _SUPER.DLL encapsulates communication port functionality (for both serial and TCP/IP ports). Files associated with this module are SERIAL.H and SOCKET.H (C++ include files), SERIAL.BAS and SOCKET.BAS (Visual Basic modules), SERIAL.INI and SOCKET.INI (default configuration data files), and SERIAL.ERR and SOCKET.ERR (error files).

- SUPERCOM.DLL, SCRS232.DLL, and SCTCPIP.DLL are supplemental communication modules required by the library.

3.2 Compiling, Linking, and Executing User Applications

3.2.1 Compiling/Linking with Visual C++

To compile a Visual C++ application, include (with #include statements) LYNXLIB.H, ISOLYNX.H, SERIAL.H, and SOCKET.H header files to project source files that reference the isoLynx Data Acquisition Library. Make it possible for your application to statically link to the library by adding the LYNXW32.LIB import library into your project. (Do this in Microsoft Visual C++ with the

Project>Add To Project>Files… menu selection.) Header files and the import library file can be found on the installation CD in the CD:\LIB directory.

3.2.2 Compiling/Linking with Visual Basic

To build a Visual Basic application, simply include the LYNXLIB.BAS, ISOLYNX.BAS, SERIAL.BAS, and SOCKET.BAS module files to your project. (Do this in Microsoft Basic with the **Project>Add File…** menu selection.) The module files can be found on the installation CD in the CD:\LIB directory.

3.2.3 Executing Applications

After your application is built, several files are necessary to execute it. First and foremost, the DLL files (LYNXW32.DLL, _ISOLYNX.DLL, _SUPER.DLL, SUPERCOM.DLL, SCRS232.DLL, and SCTCPIP.DLL) need to be installed in an appropriate directory. DLLs can always be placed in the same directory as the executable. See *dLibOpen* for information on installing DLLs in other directories.

A script file must also be specified to the library. The script file contains communication settings and channel configuration information used by the library. If you don't have a script file, one can be created from the .INI files (LYNXLIB.INI, ISOLYNX.INI, SERIAL.INI, and SOCKET.INI) by specifying an empty filename to the library and by selecting the appropriate options to *dComOpen* and *dIoOpen*. Install the .INI files in the *datpath* directory specified to *dLibOpen*.

Finally, if you would like to decode errors with *dLibError*, the .ERR files (LYNXLIB.ERR, ISOLYNX.ERR, SERIAL.ERR, and SOCKET.ERR) should be installed in an appropriate directory. Install these files in the *datpath* directory specified to *dLibOpen*.

3.3 isoLynx API

The API functions and data types are divided into four main groups: Library, Communications, I/O, and File. Each is described by their respective sections below.

3.3.1 Error Codes

All API functions return an integer error code. 0 indicates no errors or warnings, a positive integer indicates a critical error, and a negative integer indicates a non-critical error or warning. A complete listing of all isoLynx API error codes can be found in the API error files (LYNXLIB.ERR, ISOLYNX.ERR, SERIAL.ERR, and SOCKET.ERR).

3.3.2 Library Functions and Data Structures

3.3.2.1 dLibClose

dLibClose must be the last library function called. It cancels the API client application's registration with the library and performs cleanup activities.

Prototype:

```
long dLibClose( void );
```

Inputs:

None

Outputs:

None

See Also:

dLibOpen

3.3.2.2 dLibError

dLibError translates an API error code (the return code from any isoLynx API function) to a textual description. It does so by searching the API error files (LYNXLIB.ERR, ISOLYNX.ERR, SERIAL.ERR, SOCKET.ERR) for the error code. The API error files are text files that contain error codes and their corresponding textual descriptions. It is this description that is returned by *dLibError*.

The error files must exist in the data path specified when the library is opened (see *dLibOpen*). If not, a string indicating the error code was not found is returned (see below).

Prototype:

```
long dLibError( long error, char* text, long maxlen );
```

Inputs:

error The error code to translate.

text Pointer to empty character array. To be filled in by function (see below).

maxlen Maximum length of *text* array.

Outputs:

text Filled in by *dLibError* with a NULL-terminated string that is the textual description of the given *error* code.

If *error* is not found or if the error files were not found, *text* is filled in with this NULL-terminated string constant: “?”.

See Also:

dLibOpen

3.3.2.3 dLibLog

This function is used to log diagnostic information to the isoLynx API log file, LYNXLIB.LOG. If logging is enabled, an application can call *dLibLog* during program execution to supplement the diagnostic information logged by the library. *dLibLog* logs information regardless of the current *logmode*. See LIB_OPEN_STRUCT details for more information.

Prototype:

```
long dLibLog(char* text);
```

Inputs:

text Pointer to a NULL-terminated character string to be written to the isoLynx API log file. *dLibLog* will append time and date information before writing to the log file.

Outputs:

None

See Also:

dLibOpen

3.3.2.4 dLibOpen

This function must be called before any other library function. It registers the isoLynx API client application with the library and performs initialization activities.

Prototype:

```
long dLibOpen(LIB_OPEN_STRUCT* opendata);
```

Inputs:

opendata Pointer to an initialized LIB_OPEN_STRUCT.

Outputs:

None

See Also:

dLibClose, *dLibLog*, LIB_OPEN_STRUCT

3.3.2.5 LIB_OPEN_STRUCT

A pointer to this structure is passed to the *dLibOpen* function. This data structure contains parameters needed to open the isoLynx Data Acquisition Library.

Definition:

```
typedef struct LIB_OPEN_STRUCT
{
   long hwnd;                    // Application Window
                                 // Handle
   long flags;                   // Open Flags
   long logmode;                 // Logging Mode
   long logmaxchar;              // Max number of characters
                                 // per line of log file
   long logmaxline;              // Max number of lines per
                                 // log file
   long logmaxfile;              // Max number of log file
                                 // backups
   char datpath[MAXPATHLEN];     // Data File Path
   char libpath[MAXPATHLEN];     // Library Path

} LIB_OPEN_STRUCT;
```

Details:

hwnd Pointer to the application window making the call. May be NULL if called from a console application.

flags Library open flags. Select from:

LIBFLAG_OPEN_LOG	Select to enable logging of diagnostic information to a log file (LYNXLIB.LOG).

logmode If logging is enabled, select the log mode:

LOG_MODE_APIALL	All function calls, errors, and warnings are logged.
LOG_MODE_APIERR	Only critical errors are logged.
LOG_MODE_APIWARN	Critical errors and warnings are logged.

logmaxchar Specify the maximum number of characters per line of the log file.

logmaxline Specify the maximum number of lines in the log file.

logmaxfile Specify the maximum number of log file backups to keep. The library automatically saves old log files before creating a new log file. The most recent log file will be saved as *DATPATH*\LYNXLIB.L00, the second most recent to *DATPATH*\LYNXLIB.L01, and so on, up to *logmaxfile* backups.

datpath Specifies the directory the library will store log files (*.LOG) and the directory the library expects to find error files (*.ERR) and initialization files (*.INI).

libpath Specifies the directory the library expects to find its supplemental DLLs in.

Note: Since applications statically link to the LYNXW32.DLL, this file needs to be in a well-known directory. In other words,

the LYNXW32.DLL file should be in one of:

- The same directory as the application executable.
- The WINDOWS directory.
- The WINDOWS\SYSTEM directory.
- A directory specified in the system PATH.

All other DLLs must be located in the directory specified by *libpath*.

See Also:

dLibOpen, *dLibLog*, *dLibError*

3.3.3 Communications Functions and Data Structures

3.3.3.1 dComClose

dComClose is used to close the specified communication port. Each *dComOpen* call must be paired with a call to *dComClose*.

Prototype:

```
long dComClose( long hcom );
```

Inputs:

hcom Communication port handle obtained by *dComOpen* call.

Outputs:

None

See Also:

dComOpen

3.3.3.2 dComConfigure

dComConfigure is used to configure the specified communication port.

Prototype:

```
long dComConfigure( long hcom, long type, void* confdata );
```

Inputs:

hcom Communication port handle obtained by *dComOpen* call.

type Select one of:

COMCONF_SERIAL	Select to configure a serial port.
COMCONF_SOCKET	Select to configure a TCP/IP port.

confdata Pointer to a data type initialized with configuration data.

If *type* is COMCONF_SERIAL, points to a COMCONF_SERIAL_STRUCT.

If *type* is COMCONF_SOCKET, points to a COMCONF_SOCKET_STRUCT.

Outputs:

None

See Also:

COMCONF_SERIAL_STRUCT, COMCONF_SOCKET_STRUCT, *dComOpen*

3.3.3.3 dComInquire

The *dComInquire* function is used to get configuration data from the specified communication port.

Prototype:

```
long dComInquire( long hcom, long type, void* confdata
);
```

Inputs:

hcom Communication port handle obtained by *dComOpen* call.

type Select one of:

COMCONF_SERIAL	Select to inquire about a serial port.
COMCONF_SOCKET	Select to inquire about a TCP/IP port.

confdata Pointer to a data type to be filled in with configuration data.

If *type* is COMCONF_SERIAL, points to a COMCONF_SERIAL_STRUCT.

If *type* is COMCONF_SOCKET, points to a COMCONF_SOCKET_STRUCT.

Outputs:

confdata *confdata* is filled in by *dComInquire* with configuration data for the requested port.

See Also:

COMCONF_SERIAL_STRUCT, COMCONF_SOCKET_STRUCT, *dComOpen*

3.3.3.4 dComOpen

dComOpen is used to obtain a handle to the specified communication port. The handle obtained is passed to the *dComClose* function and all *dIoOpen* functions targeted for the specified port. Multiple Communication port handles can be opened and used concurrently.

Prototype:

```
long dComOpen( long type, void* opendata, long* hcom );
```

Inputs:

type Select one of:

COMTYPE_SERIAL	Select to open a serial port.
COMTYPE_SOCKET	Select to open a TCP/IP port.

opendata Pointer to a data type initialized with open data.

If *type* is COMTYPE_SERIAL, points to a COMOPEN_SERIAL_STRUCT.

If *type* is COMTYPE_SOCKET, points to a COMOPEN_SOCKET_STRUCT.

Outputs:

hcom If *dComOpen* is successful, *hcom* is a handle to the specified Communication port.

See Also:

COMOPEN_SERIAL_STRUCT, COMOPEN_SOCKET_STRUCT, *dComClose*

3.3.3.5 dComReceive

dComReceive is used to receive bytes from the specified communication port. Note: Use of *dComSend* and *dComReceive* should be limited to instances when none of the I/O functions can accomplish the task. This type of situation should occur rarely, if ever.

If the specified port is opened for non-blocking operation, *dComReceive* may return an ERR_COM_PENDING warning, indicating that the receive operation has started, but is not yet complete. In this case, call *dComStatus* at some later time to check for completion of the receive process. Note that the receive process will still access *recvdata* and *count*, so these variables should not be used until *dComStatus* indicates the receive operation has completed.

Prototype:

```
long dComReceive( long hcom, long length, long maxlen,
                  void* recvdata, long* count );
```

Inputs:

hcom Communication port handle obtained by *dComOpen* call.

length Number of data bytes to receive.

maxlen *recvdata* buffer size.

Outputs:

recvdata Filled in with received data bytes.

count Filled in with the number of data bytes received.

See Also:

dComOpen, *dComStatus*

3.3.3.6 dComSend

dComSend is used to send data out the specified communication port. Note: Use of *dComSend* and *dComReceive* should be limited to instances when none of the I/O functions can accomplish the task. This type of situation should occur rarely, if ever.

If the specified port is opened for non-blocking operation, *dComSend* may return an ERR_COM_PENDING warning, indicating that the send operation has started, but is not yet complete. In this case, call *dComStatus* at some later time to check for completion of the transmission. Note that the send process will still access *senddata* and *count*, so these variables should not be used until *dComStatus* indicates the send operation has completed.

Prototype:

```
long dComSend(  long hcom, long length, long maxlen,
                void* senddata, long* count );
```

Inputs:

hcom Communication device handle obtained by *dComOpen* call.

length Number of data bytes to send.

maxlen *senddata* buffer size.

senddata Buffer filled with data bytes to send.

Outputs:

count Filled in with the number of data bytes sent.

See Also:

dComOpen, *dComStatus*

3.3.3.7 dComStatus

The *dComStatus* function is used to check the status of a pending (non-blocking) send or receive operation.

Prototype:

```
long dComStatus(   long hcom, long mask, long* status
                   void* statdata, long* count );
```

Inputs:

hcom Communication device handle obtained by *dComOpen* call.

mask Select one of:

COMSTAT_SELECT_RECV	Select to receive status on a pending receive.
COMSTAT_SELECT_SEND	Select to receive status on a pending send.

Outputs:

status Indicates current state of send or receive operation. If the operation is still pending, *status* will be 0. If the operation is complete, a corresponding status bit will be set (i.e. the COMSTAT_SELECT_RECV bit will be set for receive operations, and the COMSTAT_SELECT_SEND bit will be set for send operations.)

statdata Not implemented at this time. User can specify NULL for pointer.

count Filled in with the current number of data bytes sent or received so far.

See Also:

dComOpen, *dComReceive, dComSend*

3.3.3.8 COMCONF_SERIAL_STRUCT

A pointer to this data structure may be passed to the *dComConfigure* and *dComInquire* functions. This data structure describes the configuration of a serial port.

Definition:

```
typedef struct COMCONF_SERIAL_STRUCT
{
    long flowctrl;      // Flow Control
    long baudrate;      // Baud Rate
    long rxflags;       // Receive Control Flags
    long txflags;       // Transmit Control Flags
    long rxtimeout;     // Receive Timeout
    long txtimeout;     // Transmit Timeout
    long rxterminator;  // Receive Terminator
    long txterminator;  // Transmit Terminator

} COMCONF_SERIAL_STRUCT;
```

Details:

See COMOPEN_SERIAL_STRUCT for field descriptions.

See Also:

COMOPEN_SERIAL_STRUCT, *dComConfigure*, *dComInquire*

3.3.3.9 COMCONF_SOCKET_STRUCT

A pointer to this data structure may be passed to the *dComConfigure* and *dComInquire* functions. This data structure describes the configuration of a TCP/IP port.

Definition:

```
typedef struct COMCONF_SOCKET_STRUCT
{
    long rxflags;       // Receive Control Flags
    long txflags;       // Transmit Control Flags
    long rxtimeout;     // Receive Timeout
    long txtimeout;     // Transmit Timeout
    long rxterminator;  // Receive Terminator
    long txterminator;  // Transmit Terminator

} COMCONF_SOCKET_STRUCT;
```

Details:

See COMOPEN_SOCKET_STRUCT for field descriptions.

See Also:

COMOPEN_SOCKET_STUCT, *dComConfigure*, *dComInquire*

3.3.3.10 COMOPEN_SERIAL_STRUCT

A pointer to this data structure is passed to the *dComOpen* function. This data structure describes the configuration of a serial port.

A script file may or may not be specified when opening a serial port. Each of the fields in COMOPEN_SERIAL_STRUCT (except *hwnd*, *flags*, and *script*) is also a parameter in the [Serial] section of the script file. If specified, the library will use the parameters found in the [Serial] section of the script file to attempt to open the serial port instead of using the fields in COMOPEN_SERIAL_STRUCT. See the SERIAL.INI file (on the installation CD in the CD:\LIB directory) for a default [Serial] section.

Definition:

```
typedef struct COMOPEN_SERIAL_STRUCT
{
    long hwnd;                  // Application Window
                                // Handle
    long flags;                 // Open Flags
    long comport;               // COM Port
    long flowctrl;              // Flow Control
    long baudrate;              // Baud Rate
    long parity;                // Parity
    long databits;              // Number of Data Bits
    long stopbits;              // Number of Stop Bits
    long rxbuflen;              // Receive Buffer Length
    long txbuflen;              // Transmit Buffer Length
    long rxflags;               // Receive Control Flags
    long txflags;               // Transmit Control Flags
    long rxtimeout;             // Receive Timeout
    long txtimeout;             // Transmit Timeout
    long rxterminator;          // Receive Terminator
```

```
    long txterminator;          // Transmit Terminator
    char script[MAXPATHLEN];    // Script File Path

} COMOPEN_SERIAL_STRUCT;
```

Details:

hwnd Pointer to the application window making the call. May be NULL if called from a console application.

flags Open flags. Select from:

COMFLAG_SCRIPT	Set this flag to use a script file to open a Serial COM port. See *script* for more details. If this flag is set, the remaining fields (except for *script*) do not need to be initialized.
COMFLAG_SCRIPTRESET	Setting this flag instructs the library to reset the [Serial] section of the given script file to the default values found in SERIAL.INI. The COMFLAG_SCRIPT flag must be set if this flag is set.

comport Select the COM port (1 for COM1, 2 for COM2, etc.)

flowctrl isoLynx does not support hardware or software handshaking. Select one of:

SERIAL_FLOW_NONE	No flow control.
SERIAL_FLOW_RS485	Select if using a 2-wire RS-485 serial interface.

baudrate Baud rate. Select one of: 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200.

parity Parity is not supported. Set to SERIAL_PARITY_NONE.

databits Number of data bits. Set to 8.

stopbits Number of stop bits. Set to 1.

rxbuflen Receive Buffer length. Suggested value: 2048

txbuflen Transmit Buffer length. Suggested value: 2048

rxflags Receive control flags. Select from:

COMFLAG_NOBLOCK	If set, specifies non-blocking receive operations (see *dComReceive*). The default is to not set this flag.
COMFLAG_TERMINATOR	If set wait for *rxterminator* to signal completion of an incoming data stream. This flag must be set to communicate with an isoLynx system.
COMFLAG_RXFLUSH	Set to flush the internal receive buffer before receiving data. The default is to not set this flag.
COMFLAG_TXFLUSH	Set to flush the internal send buffer before receiving data. The default is to not set this flag.

txflags Transmit control flags. See *rxflags*.

COMFLAG_NOBLOCK	If set, specifies non-blocking send operations (see *dComSend*). The default is to not set this flag.
COMFLAG_TERMINATOR	If set, append *txterminator* at the end of every data stream sent. This flag must be set to communicate with an isoLynx system.
COMFLAG_RXFLUSH	Set to flush the internal receive buffer before sending data. The default is to set this flag.
COMFLAG_TXFLUSH	Set to flush the internal send buffer before sending data. The default is to not set this flag.
COMFLAG_TXECHO	If set, instructs the library to expect to receive back any data sent out the communication port. Ignore this echoed data. Set this flag when communicating over a network that echoes data back.

rxtimeout Receive timeout in milliseconds.

txtimeout Transmit timeout in milliseconds.

rxterminator Specifies response termination character. This is the character that indicates the completion of an incoming data stream. Always specify COMTERM_CR.

COMTERM_CR	Carriage Return indicates end of reception.

txterminator Specifies command termination character. This character will be appended by the library to the end of each outgoing data stream. Always specify COMTERM_CR (see above).

script If the COMFLAG_SCRIPT flag is set, this field is the path to the script file to use in opening the serial COM port. The file must contain “name=value” pairs in a [Serial] section for each field of the COMOPEN_SERIAL_STRUCT (except *hwnd*, *flags*, and *script*). Refer to the [Serial] section of the SERIAL.INI file for an example.

See Also:

dComOpen

3.3.3.11 COMOPEN_SOCKET_STRUCT

A pointer to this data structure is passed to the *dComOpen* function. This data structure describes the configuration of a TCP/IP port.

A script file may or may not be specified when opening a TCP/IP port. Each of the fields in COMOPEN_SOCKET_STRUCT (except *hwnd*, *flags*, and *script*) is also a parameter in the [Socket] section of the script file. If specified, the library will use the parameters found in the [Socket] section of the script file to attempt to open the TCP/IP port instead of using the fields in COMOPEN_SOCKET_STRUCT. See the SOCKET.INI file (on the installation CD in the CD:\LIB directory) for a default [Socket] section.

Definition:

```
typedef struct COMOPEN_SOCKET_STRUCT
{
    long hwnd;                  // Application Window
                                // Handle
    long flags;                 // Open Flags
    long rxflags;               // Receive Control Flags
    long txflags;               // Transmit Control Flags
    long rxtimeout;             // Receive Timeout
    long txtimeout;             // Transmit Timeout
    long rxterminator;          // Receive Terminator
    long txterminator;          // Transmit Terminator
    long tcpport;               // TCP Server Port Number
    long timeout;               // TCP Connect Timeout
    long server[MAXSERVLEN];    // TCP Server IP Address
    char script[MAXPATHLEN];    // Script File Path

} COMOPEN_SOCKET_STRUCT;
```

Details:

hwnd Pointer to the application window making the call. May be NULL if called from a console application.

flags Open flags. Select from:

COMFLAG_SCRIPT	Set this flag to use a script file to open a TCP/IP port. See *script* for more details. If this flag is set, the remaining fields (except for *script*) do not need to be initialized.
COMFLAG_SCRIPTRESET	Setting this flag instructs the library to reset the [Socket] section of the given script file to the default values found in SOCKET.INI. The COMFLAG_SCRIPT flag must be set if this flag is set.

rxflags Receive control flags. Select from:

COMFLAG_NOBLOCK	If set, specifies non-blocking receive operations (see *dComReceive*). The default is to not set this flag.
COMFLAG_TERMINATOR	If set wait for *rxterminator* to signal completion of an incoming data stream. This flag must be set to communicate with an isoLynx system.
COMFLAG_RXFLUSH	Set to flush the internal receive buffer before receiving data. The default is to not set this flag.
COMFLAG_TXFLUSH	Set to flush the internal send buffer before receiving data. The default is to not set this flag.

txflags Transmit control flags.

COMFLAG_NOBLOCK	If set, specifies non-blocking send operations (see *dComSend*). The default is to not set this flag.
COMFLAG_TERMINATOR	If set, append *txterminator* at the end of every data stream sent. This flag must be set to communicate with an isoLynx system.
COMFLAG_RXFLUSH	Set to flush the internal receive buffer before sending data. The default is to set this flag.
COMFLAG_TXFLUSH	Set to flush the internal send buffer before sending data. The default is to not set this flag.
COMFLAG_TXECHO	If set, instructs the library to expect to receive back any data sent out the communication port. Ignore this echoed data. Set this flag when communicating over a network that echoes data back.

rxtimeout Receive timeout in milliseconds.

txtimeout Transmit timeout in milliseconds.

rxterminator Specifies response termination character. This is the character that indicates the completion of an incoming data stream. Always specify COMTERM_CR.

COMTERM_CR	Carriage Return indicates end of reception.

txterminator Specifies command termination character. This character will be appended by the library to the end of each outgoing data stream. Always specify COMTERM_CR (see above).

tcpport TCP server port number (e.g. 9000).

timeout TCP connect timeout in milliseconds.

server NULL terminated string indicating TCP server IP address (e.g. “255.255.255.0”.

script If the COMFLAG_SCRIPT flag is set, this field is the path to the script file to use in opening the TCP/IP port. The file must contain “name=value” pairs in a [Socket] section for each field of COMOPEN_SOCKET_STRUCT (except *hwnd*, *flags*, and *script*). Refer to the [Socket] section of the SOCKET.INI file for an example.

See Also:

dComOpen

3.3.4 IO Functions and Data Structures

3.3.4.1 dIoClose

dIoClose is used to release an isoLynx device handle. Each *dIoOpen* call must be paired with a call to *dIoClose*.

Prototype:

```
long dIoClose( long hio );
```

Inputs:

hio isoLynx device handle obtained by *dIoOpen* call.

Outputs:

None

See Also:

dIoOpen

3.3.4.2 dIoConfigure

dIoConfigure is used to configure an isoLynx system and its I/O channels.

Prototype:

```
long dIoConfigure(  long hio, long count, long type,
                    void* attrdata );
```

Inputs:

hio isoLynx device handle obtained by *dIoOpen* call.

count Number of data structures in the *attrdata* list.

type Specifies what to configure. Select one of:

IOATTR_CHANNEL_CNF	Use to configure I/O channels (e.g. input or output, channel tag, channel units, etc.). *attrdata* must point to an array of IOATTR_CHANNEL_CNF_STRUCT data types.
IOATTR_ISOLYNX_INTERFACE	Use to configure the isoLynx communication interface (e.g. RS-232/RS-485/Ethernet, baud rate, etc.). *attrdata* must point to an array of IOATTR_ISOLYNX_INTERFACE_STRUCT data types.
IOATTR_ISOLYNX_AIOPTION	Use to configure analog input channel parameters (e.g. range, gain, sampling weight, etc.). *attrdata* must point to an array of IOATTR_ISOLYNX_AIOPTION_STRUCT data types.
IOATTR_ISOLYNX_AOOPTION	Use to configure analog output channel parameters (e.g. range, gain, default value, etc.). *attrdata* must point to an array of

IOATTR_ISOLYNX_AOOPTION_STRUCT data types.

IOATTR_ISOLYNX_DIOPTION	Use to configure digital input channel parameters (e.g. mode, etc.). *attrdata* must point to an array of IOATTR_ISOLYNX_DIOPTION_STRUCT data types.
IOATTR_ISOLYNX_DOOPTION	Use to configure digital output channel parameters (e.g. mode, default value, etc.). *attrdata* must point to an array of IOATTR_ISOLYNX_DOOPTION_STRUCT data types.
IOATTR_ISOLYNX_NETOPTION	Use to configure Ethernet options (e.g. TCP server port, IP address, etc.). *attrdata* must point to an array of IOATTR_ISOLYNX_NETOPTION_STRUCT data types.

attrdata Points to an array of data structures of type specified by the *type* input parameter (see above).

Outputs:

None

See Also:

dIoOpen, *dIoInquire*,
IOATTR_CHANNEL_CNF_STRUCT,
IOATTR_ISOLYNX_INTERFACE_STRUCT,
IOATTR_ISOLYNX_AIOPTION_STRUCT,
IOATTR_ISOLYNX_AOOPTION_STRUCT,
IOATTR_ISOLYNX_DIOPTION_STRUCT,
IOATTR_ISOLYNX_DOOPTION_STRUCT,
IOATTR_ISOLYNX_NETOPTION_STRUCT

3.3.4.3 dIoInquire

dIoInquire is used to get configuration information from an isoLynx and its I/O channels.

Prototype:

```
long dIoInquire(   long hio, long count, long type,
                   void* attrdata );
```

Inputs:

hio isoLynx device handle obtained by *dIoOpen* call.

count Number of items to get inquiry data from.

type Specifies what to get inquiry data from. Select one of:

IOATTR_CHANNEL_CNF	Use to retrieve I/O channel configuration (e.g. input or output, channel tag, channel units, etc.). *attrdata* must point to an array of IOATTR_CHANNEL_CNF_STRUCT data types.
IOATTR_ISOLYNX_INTERFACE	Use to retrieve isoLynx communication interface settings (e.g. RS-232/RS-485/Ethernet, baud rate, etc.). *attrdata* must point to an array of IOATTR_ISOLYNX_INTERFACE_STRUCT data types.
IOATTR_ISOLYNX_AIOPTION	Use to retrieve analog input channel parameters (e.g. range, gain, sampling weight, etc.). *attrdata* must point to an array of IOATTR_ISOLYNX_AIOPTION_STRUCT data types.
IOATTR_ISOLYNX_AOOPTION	Use to retrieve analog output channel parameters

	(e.g. range, gain, default value, etc.). *attrdata* must point to an array of IOATTR_ISOLYNX_AOOPTION_STRUCT data types.
IOATTR_ISOLYNX_DIOPTION	Use to retrieve digital input channel parameters (e.g. mode, etc.). *attrdata* must point to an array of IOATTR_ISOLYNX_DIOPTION_STRUCT data types.
IOATTR_ISOLYNX_DOOPTION	Use to retrieve digital output channel parameters (e.g. mode, default value, etc.). *attrdata* must point to an array of IOATTR_ISOLYNX_DOOPTION_STRUCT data types.
IOATTR_ISOLYNX_NETOPTION	Use to retrieve Ethernet settings (e.g. TCP server port, IP address, etc.). *attrdata* must point to an array of IOATTR_ISOLYNX_NETOPTION_STRUCT data types.

Outputs:

attrdata Points to an array of data structures of type specified by the *type* input. Filled in with configuration information by the *dIoInquire* function.

See Also:

dIoOpen, *dIoConfigure*,
IOATTR_CHANNEL_CNF_STRUCT,
IOATTR_ISOLYNX_INTERFACE_STRUCT,
IOATTR_ISOLYNX_AIOPTION_STRUCT,
IOATTR_ISOLYNX_AOOPTION_STRUCT,
IOATTR_ISOLYNX_DIOPTION_STRUCT,
IOATTR_ISOLYNX_DOOPTION_STRUCT,
IOATTR_ISOLYNX_NETOPTION_STRUCT

3.3.4.4 dIoOpen

The *dIoOpen* function is used to obtain a handle to an isoLynx system. This handle is passed to the *dIoClose* function and all I/O functions targeted for the corresponding isoLynx. Multiple device handles can be opened and used concurrently.

Note: This function must be called after *dComOpen*.

Prototype:

```
long dIoOpen( long type, void* opendata, long* hio );
```

Inputs:

type Specifies the I/O device type to open. Set to:

IOTYPE_ISOLYNX	Open a handle to an isoLynx system.

opendata Pointer to an initialized IOOPEN_ISOLYNX_STRUCT data structure.

Outputs:

hio isoLynx device handle.

See Also:

dIoClose, IOOPEN_ISOLYNX_STRUCT

3.3.4.5 dIoRead

dIoRead is used to read data from the I/O channels of the specified isoLynx system.

Data can be read in raw counts or in floating-point format. (See the IOFLAG_FLOAT flag in IOCTRL_DEFAULT_STRUCT.) If counts are read, the 16-bit result of the latest A/D conversion is returned without modification by the library. If floating-point data is read, the library applies a formula, using each channel's *gain* and *offset* parameters, to the counts before returning it to the caller. The formula used is:

floating-point data = (counts * *gain*) + *offset*

A channel's *gain* and *offset* parameters may be set with the *dIoConfigure* command.

This function may be called in blocking or non-blocking mode. In blocking mode, *dIoRead* will not return to the caller until the command response from the isoLynx system has been received. This is the default mode of operation. In non-blocking mode, *dIoRead* returns immediately with an ERR_IO_PENDING warning. The command response from the isoLynx system has been received when *dIoRead* returns something other than ERR_IO_PENDING (either a different error/warning or ERR_SUCCESS).

This command can be used to read data from device output channels as well as input channels. To read data from an output channel means to retrieve from the library the value last written. No commands are sent to the isoLynx system for this type of read. (See the IOFLAG_CACHE flag in IOCTRL_DEFAULT_STRUCT.)

Prototype:

```
long dIoRead(  long hio, long type, void* readctrl,
               void* readdata, void* readstat );
```

Inputs:

hio isoLynx device handle obtained by *dIoOpen* call.

type Set to IOCTRL_DEFAULT.

readctrl Pointer to an initialized IOCTRL_DEFAULT_STRUCT. This data type contains the read controls. See IOCTRL_DEFAULT_STRUCT for details on the internals of this data structure.

Outputs:

readdata Pointer to a buffer for data to be read into. If the IOFLAG_FLOAT flag is set (see IOCTRL_DEFAULT_STRUCT), this field will be filled in with *double*-precision floating point values. If the IOFLAG_FLOAT flag is not set, this field will be filled in with *long* integer values (counts).

readstat Points to an array of *unsigned long* integers. Filled in with isoLynx API error codes for each channel. If the IOFLAG_GROUP flag was set (see IOCTRL_DEFAULT_STRUCT), all channels will have the same error code. Use *dLibError* to decode.

See Also:

dIoOpen, *dIoWrite*, *dLibError*, IOCTRL_DEFAULT_STRUCT

3.3.4.6 dIoReset

dIoReset is used to reset panels of an isoLynx system.

Prototype:

```
long dIoReset( long hio, long reset );
```

Inputs:

hio I/O device handle obtained by *dIoOpen* call.

reset Reset mode. Select from:

IORESET_DEFAULT	Issue a standard reset command to all panels.
IOFLAG_RESET_FACTORY	Reset the channel configuration of all panels to the factory default (i.e. no channels configured).
IORESET_ISOLYNX_DEFAULT	Send a standard reset command to specified panels. OR any combination of panel constants (see below) with this value to reset the corresponding panels.
IORESET_ISOLYNX_FACTORY	Reset the channel configuration on selected panels to the factory default (i.e. no channels configured). OR any combination of panel constants (see below) with this value to reset the channel configuration of corresponding panels.

Panel Constants:

IORESET_ISOLYNX_PANEL_A0	All Analog I/O Panels.

IORESET_ISOLYNX_PANEL_D0	Digital I/O Panel 0.
IORESET_ISOLYNX_PANEL_D1	Digital I/O Panel 1.
IORESET_ISOLYNX_PANEL_D2	Digital I/O Panel 2.
IORESET_ISOLYNX_PANEL_D3	Digital I/O Panel 3.
IORESET_ISOLYNX_PANEL_D4	Digital I/O Panel 4.
IORESET_ISOLYNX_PANEL_D5	Digital I/O Panel 5.
IORESET_ISOLYNX_PANEL_D6	Digital I/O Panel 6.
IORESET_ISOLYNX_PANEL_D7	Digital I/O Panel 7.

Outputs:

None

See Also:

dIoOpen

3.3.4.7 dIoWrite

The *dIoWrite* function is used to write data to I/O channels of the specified isoLynx system.

Data can be written as raw counts or in floating-point format. (See the IOFLAG_FLOAT flag in IOCTRL_DEFAULT_STRUCT.) If counts are written, the 16-bit value for a single channel is passed to the D/A converter without modification by the library. If floating-point data is written, the library applies a formula, using each channel's *gain* and *offset* parameters, to the raw counts before writing it to the D/A converter. The formula used is:

counts = (floating-point data – *offset*) / *gain*

A channel's *gain* and *offset* parameters may be set with the *dIoConfigure* command.

This function may be called in blocking or non-blocking mode. In blocking mode, *dIoWrite* will not return to the caller until the command response from the isoLynx system has been received. This is the default mode of operation. In non-blocking mode, *dIoWrite* returns immediately with an ERR_IO_PENDING warning. The command response from the isoLynx system has been received when *dIoWrite* returns something other than ERR_IO_PENDING (either a different error/warning or ERR_SUCCESS).

Unlike the *dIoRead* command, data may only be written to channels configured as outputs.

Prototype:

```
long dIoWrite(  long hio, long type,
                void* writectrl,
                void* writedata,
                void* writestat );
```

Inputs:

hio isoLynx device handle obtained by *dIoOpen* call.

type Set to IOCTRL_DEFAULT.

writectrl Pointer to an initialized IOCTRL_DEFAULT_STRUCT. This data structure contains the write controls. See IOCTRL_DEFAULT_STRUCT for details on the internals of this data structure.

writedata Pointer to a buffer of data to write. If the IOFLAG_FLOAT flag is set (see IOCTRL_DEFAULT_STRUCT), this field should point to an array of *double*-precision floating-point values. If the IOFLAG_FLOAT flag is not set, this field should point to an array of *long* integer values (counts).

Outputs:

writestat Points to an array of *unsigned long* integers. Filled in with isoLynx API error codes for each channel. If the IOFLAG_GROUP flag was set (see IOCTRL_DEFAULT_STRUCT), all channels will have the same error code. Use *dLibError* to decode.

See Also: *dIoOpen*, *dIoRead*, IOCTRL_DEFAULT_STRUCT

3.3.4.8 IOATTR_CHANNEL_CNF_STRUCT

A pointer to this data type may be passed into the *dIoConfigure* and *dIoInquire* functions. This data structure describes the configuration of a single isoLynx channel.

Definition:

```
typedef struct IOATTR_CHANNEL_CNF_STRUCT
{
    long panel;              // I/O Panel
    long channel;            // I/O Channel
    long iotype;             // I/O Type
    char tag[MAXTAGLEN];     // I/O Channel Tag
    char units[MAXTAGLEN];   // I/O Channel Units
    void* options;           // Reserved

} IOATTR_CHANNEL_CNF_STRUCT;
```

Details:

panel Specifies the I/O panel the channel is on.

For Analog I/O Panels, set to IOPANEL_ISOLYNX_AIO + n (n = 0, 1, 2, or 3). n = 0 is the isoLynx Analog I/O Base Unit.

For Digital I/O Panels, set to IOPANEL_ISOLYNX_DIO + n (n = 0, 1, 2, 3, 4, 5, 6, or 7).

channel Specifies the channel number (0-15).

iotype Specifies the channel type. Select one of:

IOTYPE_NONE	Channel not configured.
IOTYPE_ISOLYNX_AI	Channel is Analog Input.
IOTYPE_ISOLYNX_AO	Channel is Analog Output.
IOTYPE_ISOLYNX_DI	Channel is Digital Input.
IOTYPE_ISOLYNX_DO	Channel is Digital Output.

tag NULL-terminated string that specifies a name for the channel (e.g. “Channel 0”).

units NULL-terminated string that specifies engineering units for the channel (e.g. “Volts”).

options Reserved. Set to NULL when calling *dComConfigure*.

See Also:

dComConfigure, *dComInquire*

3.3.4.9 IOATTR_ISOLYNX_AIOPTION_STRUCT

A pointer to this data type may be passed into the *dIoConfigure* and *dIoInquire* functions. This data structure describes the configuration of a single analog input channel.

Definition:

```
typedef struct IOATTR_ISOLYNX_AIOPTION_STRUCT
{
    long panel;         // I/O Panel
    long channel;       // I/O Channel
    long weight;        // Averaging Weight
    long range;         // Range
    doube gain;         // Gain
    double offset;      // Offset
    long reserved0;     // Reserved
    long reserved1;     // Reserverd
    long reserved2;     // Reserved
    long reserved3;     // Reserved

} IOATTR_ISOLYNX_AIOPTION_STRUCT;
```

Details:

panel Specifies the I/O panel the channel is on.

For analog panels, set to IOPANEL_ISOLYNX_AIO + n (n = 0, 1, 2, or 3). n = 0 is the isoLynx Analog I/O Base Unit.

channel Specifies the channel number (0-15).

range Specifies the channel's range. Currently, only one range is supported:

AIO_RANGE_BIPOLAR_10	Input range is +/- 10 volts.

weight Sample weight used in calculating the channel's running average. Valid values for weight are the powers of 2: n = 0, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, and 16384. A sample weight of 0 means no average is calculated for the channel.

gain Channel gain (default = 1.0). Used in calculation of floating-point data (see *dIoRead* and *dIoWrite*).

offset Channel offset (default = 0.0). Used in calculation of floating-point data (see *dIoRead* and *dIoWrite*).

See Also:

dIoConfigure, *dIoInquire, dIoRead, dIoWrite*

3.3.4.10 IOATTR_ISOLYNX_AOOPTION_STRUCT

A pointer to this data type may be passed into the *dIoConfigure* and *dIoInquire* functions. This data structure describes the configuration of a single analog output channel.

Definition:

```
typedef struct IOATTR_ISOLYNX_AOOPTION_STRUCT
{
    long panel;          // I/O Panel
    long channel;        // I/O Channel
    long initial;        // Initial Output Value
    long range;          // Range
    doube gain;          // Gain
    double offset;       // Offset
    long reserved0;      // Reserved
    long reserved1;      // Reserved
    long reserved2;      // Reserved
    long reserved3;      // Reserved

} IOATTR_ISOLYNX_AOOPTION_STRUCT;
```

Details:

panel Specifies the I/O panel the channel is on.

For analog panels, set to IOPANEL_ISOLYNX_AIO + n (n = 0, 1, 2, or 3). n = 0 is the isoLynx Analog I/O Base Unit.

channel Specifies the channel number (0-15).

initial Specifies the channel's initial output value, in counts. The output gets set to this value on device reset and on configuration of the channel to an analog output.

range Specifies the channel's range. Currently, only one range is supported:

AIO_RANGE_BIPOLAR_10	Output range is +/- 10 volts.

gain Channel gain (default = 1). Used in converting floating-point data to raw data (see *dIoRead* and *dIoWrite*).

offset Channel offset (default = 0.0). Used in converting floating-point data to raw data (see *dIoRead* and *dIoWrite*).

See Also:

dIoConfigure, *dIoInquire, dIoRead, dIoWrite*

3.3.4.11 IOATTR_ISOLYNX_DIOPTION_STRUCT

A pointer to this data type may be passed into the *dIoConfigure* and *dIoInquire* functions. This data structure describes the configuration of a single digital input channel.

Definition:

```
typedef struct IOATTR_ISOLYNX_DIOPTION_STRUCT
{
    long panel;       // I/O Panel
    long channel;     // I/O Channel
    long reserved0;   // Unsupported
    long reserved1;   // Reserved
    long reserved2;   // Reserved
    long reserved3;   // Reserved

} IOATTR_ISOLYNX_DIOPTION_STRUCT;
```

Details:

panel Specifies the I/O panel the channel is on.

For digital panels, set to IOPANEL_ISOLYNX_DIO + n (n = 0, 1, 2, 3, 4, 5, 6, or 7).

channel Specifies the channel number (0-15).

mode TBD. Unsupported at this time.

See Also:

dIoConfigure, *dIoInquire, dIoRead, dIoWrite*

3.3.4.12 IOATTR_ISOLYNX_DOOPTION_STRUCT

A pointer to this data type may be passed into the *dIoConfigure* and *dIoInquire* functions. This data structure describes the configuration of a single digital output channel.

Definition:

```
typedef struct IOATTR_ISOLYNX_DOOPTION_STRUCT
{
    long panel;       // I/O Panel
    long channel;     // I/O Channel
    long initial;     // Initial Value
    long reserved0;   // Reserved
    long reserved1;   // Reserved
    long reserved2;   // Reserved
    long reserved3;   // Reserved

} IOATTR_ISOLYNX_DOOPTION_STRUCT;
```

Details:

panel Specifies the I/O panel the channel is on.

For digital panels, set to IOPANEL_ISOLYNX_DIO + n (n = 0, 1, 2, 3, 4, 5, 6, or 7).

channel Specifies the channel number (0-15).

initial The channel's initial value (1 or 0). This is the value the channel gets set to on device reset and on configuration of the channel to a digital output.

mode TBD. Unsupported at this time.

See Also:

dIoConfigure, *dIoInquire, dIoRead, dIoWrite*

3.3.4.13 IOATTR_ISOLYNX_INTERFACE_STRUCT

A pointer to this data type may be passed into the *dIoConfigure* and *dIoInquire* functions. This data structure describes the communications interface of the isoLynx system.

Note that if the interface is changed (with *dIoConfigure*), the host port may need to be reconfigured to match the new communication settings. That is, the isoLynx will respond to the *dIoConfigure* command using the old communication settings, but will process all subsequent commands with the new settings. *dComConfigure* should be called immediately after *dIoConfigure* to update the communication settings of the host port.

Definition:

```
typedef struct IOATTR_ISOLYNX_INTERFACE_STRUCT
{
    long address;     // I/O Panel
    long iftype;      // Interface Type
    long baudrate;    // Data Rate
    long options;     // Interface Options

} IOATTR_ISOLYNX_INTERFACE_STRUCT;
```

Details:

address Address of the isoLynx system (0-15).

iftype The isoLynx system's communication interface. Select from one of:

IFTYPE_ISOLYNX_RS232	Interface is RS-232.
IFTYPE_ISOLYNX_RS485_2	Interface is RS-485 2-wire.
IFTYPE_ISOLYNX_RS485_4	Interface is RS-485 4-wire.
IFTYPE_ISOLYNX_ETHERNET	Interface is Ethernet.

baudrate Communications data rate. Select from 1200, 2400, 4800, 9600, 19200, 38400, 57600, and 115200 (bps). This field is undefined if *iftype* is IFTYPE_ISOLYNX_ETHERNET.

options Interface options. Select from one of:

IFOPTN_ISOLYNX_NONE	No options. Select if *iftype* is RS-232 or Ethernet.
IFOPTN_ISOLYNX_RS485_PT2PTECHO	RS-485 point-to-point with echo. The 'with echo' qualifier instructs the library to expect any transmitted character to be received again and to ignore these echoed characters. An isoLynx system will not echo received characters back to the host, but an RS-232 to RS-485 converter might.
IFOPTN_ISOLYNX_RS485_MULTIECHO	RS-485 multi-drop with echo. The 'with echo' qualifier instructs the library to expect any transmitted character to be received again and to ignore these echoed characters. An isoLynx system will not echo received characters back to the host, but an RS-232 to RS-485 converter might.
IFOPTN_ISOLYNX_RS485_PT2PTAUTO	RS-485 point-to-point without echo. The

	'without echo' qualifier instructs the library to treat every received character as part of a command response.
IFOPTN_ISOLYNX_RS485_MULTIAUTO	RS-485 multi-drop without echo. The 'without echo' qualifier instructs the library to treat every received character as part of a command response.

See Also:

dIoConfigure, *dIoInquire*

3.3.4.14 IOATTR_ISOLYNX_NETOPTION_STRUCT

A pointer to this data type may be passed into the *dIoConfigure* and *dIoInquire* functions. This data structure describes the Ethernet configuration of the isoLynx system.

Definition:

```
typedef struct IOATTR_ISOLYNX_NETOPTION_STRUCT
{
    long flags;        // Network Option Flags
    long tcpport;      // TCP Server Port Number
    long reserved;     // Reserved

    long ipaddress[MAXISOLYNXIPLEN];    // IP Address
    char subnetmask[MAXISOLYNXIPLEN];   // Subnet Mask
    char gateway[MAXISOLYNXIPLEN];      // Gateway
    char dnsserver[MAXISOLYNXIPLEN];    // DNS Server

} IOATTR_ISOLYNX_NETOPTION_STRUCT;
```

Details:

flags Network option flags.

IOFLAG_ISOLYNX_USEDHCP	Set to use DHCP. <QUESTION: What does this mean?>

tcpport Specifies the TCP server port number (e.g. 9000).

reserved Reserved for future use.

ipaddress NULL-terminated string representing the device's IP address (e.g. "255.255.255.0").

subnetmask NULL-terminated string representing the subnet mask (e.g. "255.255.255.0").

gateway NULL-terminated string representing the gateway's IP address (e.g. "255.255.255.0").

dnsserver NULL-terminated string representing the DNS server's IP address. (e.g. "255.255.255.0").

See Also:

dIoConfigure, *dIoInquire*

3.3.4.15 IOCTRL_DEFAULT_STRUCT

A pointer to this structure is passed into the *dIoRead* and *dIoWrite* functions. It defines which I/O channels to read or write and provides instructions on how to carry out the read or write.

Definition:

```
typedef struct IOCTRL_DEFAULT_STRUCT
{
    long flags;                    // Control Flags
    long samples;                  // Number of Samples
                                   // per Channel
    long period;                   // Sample Period in
                                   // Milliseconds
    long count;                    // Number of Channels
    long panel[MAXIOCHANNEL];      // I/O Panel List
    long channel[MAXIOCHANNEL];    // I/O Channel List

} IOCTRL_DEFAULT_STRUCT;
```

Details:

flags Control flags. Select one or more from:

IOFLAG_NOBLOCK	If this flag is set, *dIoRead* and *dIoWrite* will not wait for the command to the isoLynx system complete before returning to the caller. The ERR_IO_PENDING warning will be returned if the read or write was successfully started. Call *dIoRead* or *dIoWrite* repeatedly until something other than ERR_IO_PENDING is returned (i.e. another warning or error, or ERR_SUCCESS). If this flag is not set, *dIoRead* and *dIoWrite* will wait for the command to the isoLynx system to fully complete before returning to the caller.

IOFLAG_NOSTOP — If this flag is set, and an error occurs while processing one of the channels in the *channel* list, execution continues to the next channel in the list. Error information gets stored in the *readstat* or *writestat* output fields of the *dIoRead* and *dIoWrite* commands respectively. If this flag is not set, command execution halts immediately after the error and an error code is returned.

IOFLAG_CACHE — Set this flag to read the last data written to an output channel or the last data read from an input channel. If this flag is set, no commands are issued to the isoLynx system. Instead, the data comes from the library itself. Note that the *dIoWrite* command does not support use of this flag. Setting this flag is the only way to read data from output channels.

IOFLAG_GROUP — isoLynx supports group reads and writes (i.e. one isoLynx command can read or write multiple channels). Set this flag to take advantage of this feature and read or write all channels in the channel list at once. (Note all channels must be part of the same panel, and no channel can be duplicated in the list.)

IOFLAG_FLOAT — Set this flag to read or write floating point data instead of counts. In the case of reads, some post-processing is done on the counts read from a channel to convert the data to floating-point format. Specifically, a formula is applied to the raw counts using the channel's *gain* and *offset* parameters. In the case of writes, an inverse formula is applied to the given floating-point data to convert it to counts before writing it to a channel. See *dIoConfigure* for information on configuring channel parameters such as *gain* and *offset*. See *dIoRead* and *dIoWrite* for the formulas used to convert between counts and floating-point data.

samples — Specify the number of samples per channel. Note that the size of the *readdata*, *readstat*, *writedata*, and *writestat* array inputs to *dIoRead* and *dIoWrite* are equal to (*count* * *samples*).

period — Specify the time, in milliseconds, between successive reads or writes. If the IOFLAG_GROUP flag is set, this it the time the library waits between each read or write group command. If the IOFLAG_GROUP flag is not set, this is the time the library waits between reading/writing each channel in the channel list. Set to 0 for asynchronous input or output. The resolution of this value is

approximately 50 ms.

count Number of channels to read or write. Note that the size of the *readdata*, *readstat*, *writedata*, and *writestat* array inputs to *dIoRead* and *dIoWrite* are equal to (*count* * *samples*).

panel List of panels to read or write.

For Analog I/O Panels, set to IOPANEL_ISOLYNX_AIO + n (n = 0, 1, 2, or 3). n = 0 is the isoLynx Analog I/O Base Unit.

For Digital I/O Panels, set to IOPANEL_ISOLYNX_DIO + n (n = 0, 1, 2, 3, 4, 5, 6, or 7).

There is a one-to-one correspondence between the *panel* and *channel* arrays. An item in each, taken together, describes a complete channel. For example, if *panel*[0] is 2 and *channel*[0] is 12, the first channel to read or write is channel 12 of panel 2. If *panel*[7] is 0 and *channel*[7] is 5, the eighth channel to read or write is channel 5 of panel 0.

channel List of channels to read or write. There is a one-to-one correspondence between the *panel* and *channel* arrays. An item in each, taken together, describes a complete channel. (See *panel*.)

See Also

dIoRead, dIoWrite, dIoConfigure

3.3.4.16 IOOPEN_ISOLYNX_STRUCT

A script file must be specified when *dIoOpen* is called. In addition to communication settings, the script file contains channel configuration information. (See Section 3.4 for more information on script files and channel configurations.) When *dIoOpen* is called, the channel configuration described by the script file can be ignored, reset, or used to configure the isoLynx system. See *reset* below for details.

Definition:

```
typedef struct IOOPEN_ISOLYNX_STRUCT
{
    long hwnd;          // Application Window Handle
    long hcom;          // COM Device Handle
    long reset;         // Reset Mode
    long flags;         // I/O Open Flags

    char script[MAXPATHLEN];  // Script file path.

} IOOPEN_ISOLYNX_STRUCT;
```

Details:

hwnd Handle to the application window making the call. May be NULL if called from a console application.

hcom Communication port handle obtained by a successful *dComOpen* call.

reset Reset Mode. Select one of:

IORESET_NONE	Select to not issue a reset to the isoLynx system as part of the open procedure. If a mismatch between the script file's channel configuration and the isoLynx system's channel configuration is discovered, an appropriate warning is returned by *dIoOpen.* (See ISOLYNX.H for possible warnings.)
IORESET_DEFAULT	Select to issue a standard reset to the isoLynx system as part of the open procedure. Mismatches between the script file's channel configuration and that discovered on the isoLynx are handled identical to the IORESET_NONE case.
IORESET_FACTORY	Select to reset the isoLynx system's channel configuration to the factory default (i.e. no configured channels). The channel configuration information in the script file is also reset.
IORESET_SCRIPTCONF	Select to reset the isoLynx system's channel configuration to the factory default, and then reconfigure with the channel configuration described by the script file. This is a handy way to configure an isoLynx system with one command.
IORESET_SCRIPTSYNC	Select to overwrite the script file's channel configuration with the configuration discovered on the isoLynx. This is a handy way to recreate a misplaced or deleted script file. Note: there is some information that is not stored by the isoLynx system, and thus cannot be restored in this manner. For example, channel *tag*, *units*, *gain*, and *offset* are not restorable.

flags Control flags. Select one or more from:

IOFLAG_ISOLYNX_IFRESET	Select to reset the [IsoLynx] section of the script file to the default values found in ISOLYNX.INI. Do this if the isoLynx has recently had its communication interface reset to the factory default (via a jumper on the backpanel). See the isoLynx™ Hardware User Manual for details on resetting an isoLynx's communication interface.

script NULL terminated string indicating the path to a script file.

See Also:

dIoRead, *dIoWrite*, *dIoConfigure*

3.3.5 File Functions

3.3.5.1 dFileReadF

Given a name, the *dFileReadF* function searches the given file for a "name=value" pair, and if found, returns the associated floating point value.

Prototype:

```
long dFileReadF(   char* path, char* section, char* name,
                   double* value, long* position );
```

Inputs:

path Pointer to NULL-terminated string indicating pathname of file to search.

section (Optional) Pointer to NULL-terminated string indicating File section to search.

If a file section is provided, the function will search only the section provided for a matching 'name=value' pair. File sections are defined by a bracketed section name (i.e. "[Serial]" defines the start of the Serial section).

NULL may be specified for this input if the entire file is to be searched.

name Pointer to NULL-terminated string indicating the name portion of a 'name=value' pair to search for.

Outputs:

value If a matching 'name=value' pair is found, *value* is updated with the value portion of the 'name=value' pair.

position (Optional) If a matching ‘name=value’ is found AND if a non-NULL pointer for *position* was provided by the caller, *position* is updated with the file offset to the start of the value portion of the ‘name=value’ pair.

See Also:

dFileWriteF, dFileFind

3.3.5.2 dFileReadI

Given a name, the *dFileReadI* function searches the given file for a “name=value” pair, and if found, returns the associated integer value.

Prototype:

```
long dFileReadI(   char* path, char* section, char* name,
                   long* value, long* position );
```

Inputs:

path Pointer to NULL-terminated string indicating pathname of file to search.

section (Optional) Pointer to NULL-terminated string indicating File section to search.

If a file section is provided, the function will search only the section provided for a matching ‘name=value’ pair. File sections are defined by a bracketed section name (i.e. “[Serial]” defines the start of the Serial section).

NULL may be specified for this input if the entire file is to be searched.

name Pointer to NULL-terminated string indicating the name portion of a ‘name=value’ pair to search for.

Outputs:

value If a matching ‘name=value’ pair is found, *value* is updated with the value portion of the ‘name=value’ pair.

position (Optional) If a matching ‘name=value’ is found AND if a non-NULL pointer for *position* was provided by the caller, *position* is updated with the file offset to the start of the value portion of the ‘name=value’ pair.

See Also: *dFileWriteI, dFileFind*

3.3.5.3 dFileReadS

Given a name, the *dFileReadS* function searches the given file for a "name=value" pair, and if found, returns the associated string.

Prototype:

```
long dFileReadS(   char* path, char* section,
                   char* name, long maxlen,
                   char* value, long* position );
```

Inputs:

path Pointer to NULL-terminated string indicating pathname of file to search.

section (Optional) Pointer to NULL-terminated string indicating File section to search.

If a file section is provided, the function will search only the section provided for a matching 'name=value' pair. File sections are defined by a bracketed section name (e.g. "[Serial]" defines the start of a 'Serial' section).

NULL may be specified for this input if the entire file is to be searched.

name Pointer to NULL-terminated string indicating the name portion of a 'name=value' pair to search for.

maxlen Maximum size of *value* array.

Outputs:

value If a matching ‘name=value’ pair is found, *value* is updated with the value portion of the ‘name=value’ pair. No more than *maxlen* characters will be copied from the file to *value*.

position (Optional) If a matching ‘name=value’ is found AND if a non-NULL pointer for *position* was provided by the caller, *position* is updated with the file offset to the start of the value portion of the ‘name=value’ pair.

See Also: dFileWriteS, dFileFind

3.3.5.4 dFileWriteF

Given a name and file, the *dFileWriteF* function searches the file for a "name=value" pair that has a matching name portion. If found, the function updates the value portion of the "name=value" pair with the given floating point value. If not found, the function creates a new "name=value" pair with the given name and value. Likewise, if the given file does not exist, it will be created with a "name=value" pair that matches the input parameters.

Prototype:

```
long dFileWriteF( char* path, char* section, char* name,
                  double value, long precision,
                  long* position );
```

Inputs:

path Pointer to NULL-terminated string indicating pathname of file to search. If this file is not found, it will be created.

section (Optional) Pointer to NULL-terminated string indicating file section to search.

If a file section is provided, the function will search only the section provided for a matching "name=value" pair. File sections are defined by a bracketed section name (e.g. "[Serial]" defines the start of a 'Serial' section). If the file section is not found, it will be created and the matching "name=value" pair will be added to it.

NULL may be specified for this input if the entire file is to be searched and no new section is to be created.

name Pointer to NULL-terminated string indicating the name portion of a "name=value" pair to search for.

value Value to be written to the value portion of the matching “name=value” pair.

precision Number of digits to right of decimal point.

Outputs:

position (Optional) If a non-NULL pointer was provided by the caller, *position* is updated with the file offset to the start of the value portion of the ‘name=value’ pair.

See Also:

dFileReadF, dFileFind

3.3.5.5 dFileWriteI

Given a name and file, the *dFileWriteI* function searches the file for a "name=value" pair that has a matching name portion. If found, the function updates the value portion of the "name=value" pair with the given integer value. If not found, the function creates a new "name=value" pair with the given name and value. Likewise, if the given file does not exist, it will be created with a "name=value" pair that matches the input parameters.

Prototype:

```
long dFileWriteI( char* path, char* section, char* name,
                  long value, long* position );
```

Inputs:

path Pointer to NULL-terminated string indicating pathname of file to search. If this file is not found, it will be created.

section (Optional) Pointer to NULL-terminated string indicating file section to search.

If a file section is provided, the function will search only the section provided for a matching "name=value" pair. File sections are defined by a bracketed section name (e.g. "[Serial]" defines the start of a 'Serial' section). If the file section is not found, it will be created and the matching "name=value" pair will be added to it.

NULL may be specified for this input if the entire file is to be searched and no new section is to be created.

name Pointer to NULL-terminated string indicating the name portion of a "name=value" pair to search for.

value Integer to be written to the value portion of the matching "name=value" pair.

Outputs:

position (Optional) If a non-NULL pointer was provided by the caller, *position* is updated with the file offset to the start of the value portion of the ‘name=value’ pair.

See Also:

dFileReadI, dFileFind

3.3.5.6 dFileWriteS

Given a name and file, the *dFileWriteS* function searches the file for a "name=value" pair that has a matching name portion. If found, the function updates the value portion of the "name=value" pair with the given character string. If not found, the function creates a new "name=value" pair with the given name and value. Likewise, if the given file does not exist, it will be created with a "name=value" pair that matches the input parameters.

Prototype:

```
long dFileWriteS( char* path, char* section, char* name,
                  char* value, long* position );
```

Inputs:

path Pointer to NULL-terminated string indicating pathname of file to search. If this file is not found, it will be created.

section (Optional) Pointer to NULL-terminated string indicating file section to search.

If a file section is provided, the function will search only the section provided for a matching "name=value" pair. File sections are defined by a bracketed section name (e.g. "[Serial]" defines the start of a 'Serial' section). If the file section is not found, it will be created and the matching "name=value" pair will be added to it.

NULL may be specified for this input if the entire file is to be searched and no new section is to be created.

name Pointer to NULL-terminated string indicating the name portion of a "name=value" pair to search for.

value Pointer to NULL-terminated string indicating the value to write to a matching "name=value" pair.

Outputs:

position (Optional) If a non-NULL pointer was provided by the caller, *position* is updated with the file offset to the start of the value portion of the ‘name=value’ pair.

See Also:

dFileReadS, dFileFind

3.4 Script Files

The isoLynx API stores communication parameters and channel configuration information in a script file. The script file is specified when the communication port is opened (see *dComOpen*) and again when a handle to the isoLynx system is obtained (see *dIoOpen*).

A script file is composed of several sections. The start of a section is defined by the section name enclosed in brackets (i.e. [Serial] defines the start of the Serial section). The section ends either at the start of the next section or at the end of the file. A section contains one or more "name=value" pairs, where 'name' is the name of a parameter used by the library, and 'value' is the value assigned to that parameter. The following sections are recognized by the library:

- The [Serial] section specifies the communication parameters (i.e. COM port, baud rate, etc.) the library will use when opening a serial port connected to an isoLynx system. See the SERIAL.INI file for a default [Serial] section.

- The [Socket] section specifies the communication parameters (i.e. port number, I/P address, etc.) the library will use when opening a TCP/IP port connected to an isoLynx system. See the SOCKET.INI file for a default [Socket] section.

- The [IsoLynx] section contains communication settings (i.e. isoLynx address, I/F type, baud rate, I/P address, etc) the library believes the isoLynx system is configured with. For communications to be successful, these settings should match the corresponding settings in the [Serial] and [Socket] sections. See the ISOLYNX.INI file for a default [IsoLynx] section.

- An isoLynx system's entire channel configuration can be described by a script file. This is done on a panel-by-panel basis. An [Aio#] section (# = 0-3) describes the channel configuration for a single Analog I/O Panel. A [Dio#] section (# = 0-7) describes the channel configuration for a single Digital I/O Panel. Each section, if present, contains an entry for each and every channel on the corresponding panel.

  The format for channel entries is described by the table below. Note: Spaces have been added to the channel entry formats for clarity.

Table 3.4.1 – Script File Channel Entry Formats

General Format: *channel#=iotype, tag, units, etc.*

iotype	One of AI (analog input), AO (analog output), DI (digital input), DO (digital output), or NC (not configured). The *iotype* field is mandatory for configured channels. All other fields are optional.
tag	A string that defines a name for the channel (if any).
units	A string that defines units for the channels (if any).
etc.	Remaining parameters are different for each *iotype*. See below.

Analog Inputs: *channel#=iotype, tag, units, weight, range, gain, offset*

weight	Sampling weight for calculating that running average.
range	Reserved. Should always be 0 if specified
gain	Value used by library when calculating floating-point data. (See *dIoRead*.) Default *gain* is 1.0.
offset	Value used by library when calculating floating-point data. (See *dIoRead*.) Default *offset* is 0.0.

Analog Outputs: *channel#=iotype, tag, units, initial, range, gain, offset*

initial	Channel's initial value on reset or power-on of the isoLynx. If specified, should be in the range of -32768 to 32768 (the decimal equivalent of 16-bits).
range	Reserved. Should always be 0 if specified.
gain	Value used by library when calculating floating-point data. (See *dIoWrite*.) Default *gain* is 1.0.
offset	Value used by library when calculating floating-point data. (See *dIoWrite*.) Default *offset* is 0.0.

Digital Inputs: *channel#=iotype, tag, units*

No type specific parameters.

Digital Outputs: *channel#=iotype, tag, units, initial*

initial	Channel's initial value on reset or power-on of the isoLynx. If specified, should be 1 or 0.

3.5 Sample Applications

A set of sample applications is provided to demonstrate usage of the isoLynx Data Acquisition Library. Instructions for running the applications are given by the following section.

Source code for each of the sample applications provides a starting point for development and provides examples of use. Visual C++ and Visual Basic source files can be found on the installation CD, in the CD:\SAMPLES\VC and CD:\SAMPLES\VB directories respectively.

3.5.1 Running the Samples

The following steps describe how to run the sample applications. It is assumed that you already have a powered-on isoLynx system connected to the host system. If this is not the case, see the isoLynx™ Hardware User Guide for instructions on installing an isoLynx system.

1. Create the C:\PROGRAM FILES\DATAFORTH\ISOLYNX folder on your local hard drive. From the installation CD-ROM, copy the CD:\DEMO folder and all of its contents to the folder you just created.

2. Run any one of the samples by executing the appropriate .EXE file from your local directory. You may wish to run CONFIG.EXE first to configure the communication interface and channel settings.

3.5.2 Establishing a Connection

The first step after executing any one of the samples is to connect with the attached isoLynx system. Connecting is the act of establishing a communication session between the application and the isoLynx system. The process of connecting is initiated by clicking the **Connect…** button in the upper-right portion of the window.

<Screen shots>

Clicking **Connect…** will bring up the **isoLynx Connect** dialog box. The script file the sample will use (SAMPDATA.DAT) is displayed. This file holds several communication parameters (interface type, baud rate, etc.), which will be used by the application in attempting to establish a communication session with the isoLynx system. A different script file may be chosen if desired. In most cases, however, accepting the default script file (SAMPDATA.DAT) is appropriate.

If the **Use factory default COM settings** box is checked, the contents of the script file are ignored when attempting to establish a communication session. Instead, the default communication parameters found in SERIAL.INI, SOCKET.INI, and/or ISOLYNX.INI are used. Once a communication session is successfully established, the selected script file will have its communication parameters overwritten with these default values.

If a communication session is successfully established, the **isoLynx Connect** dialog box will disappear and the sample will become active (i.e. controls will change from grayed-out to active). If unsuccessful, an appropriate error message will be displayed, and the sample will remain inactive (i.e. controls will remain grayed-out). If you are unsure of the isoLynx system's current communication settings or are having trouble establishing a connection, it is possible to reset the isoLynx system's communication settings to the factory default values. (See the isoLynx™ Hardware User Manual for instructions how.) It should then be possible to connect by checking the **Use factory default COM settings** box in the **isoLynx Connect** dialog window.

3.5.3 Configuration Sample

Once connected, the Configuration Sample may be used to change the communication interface, configure I/O channels, and modify channel options.

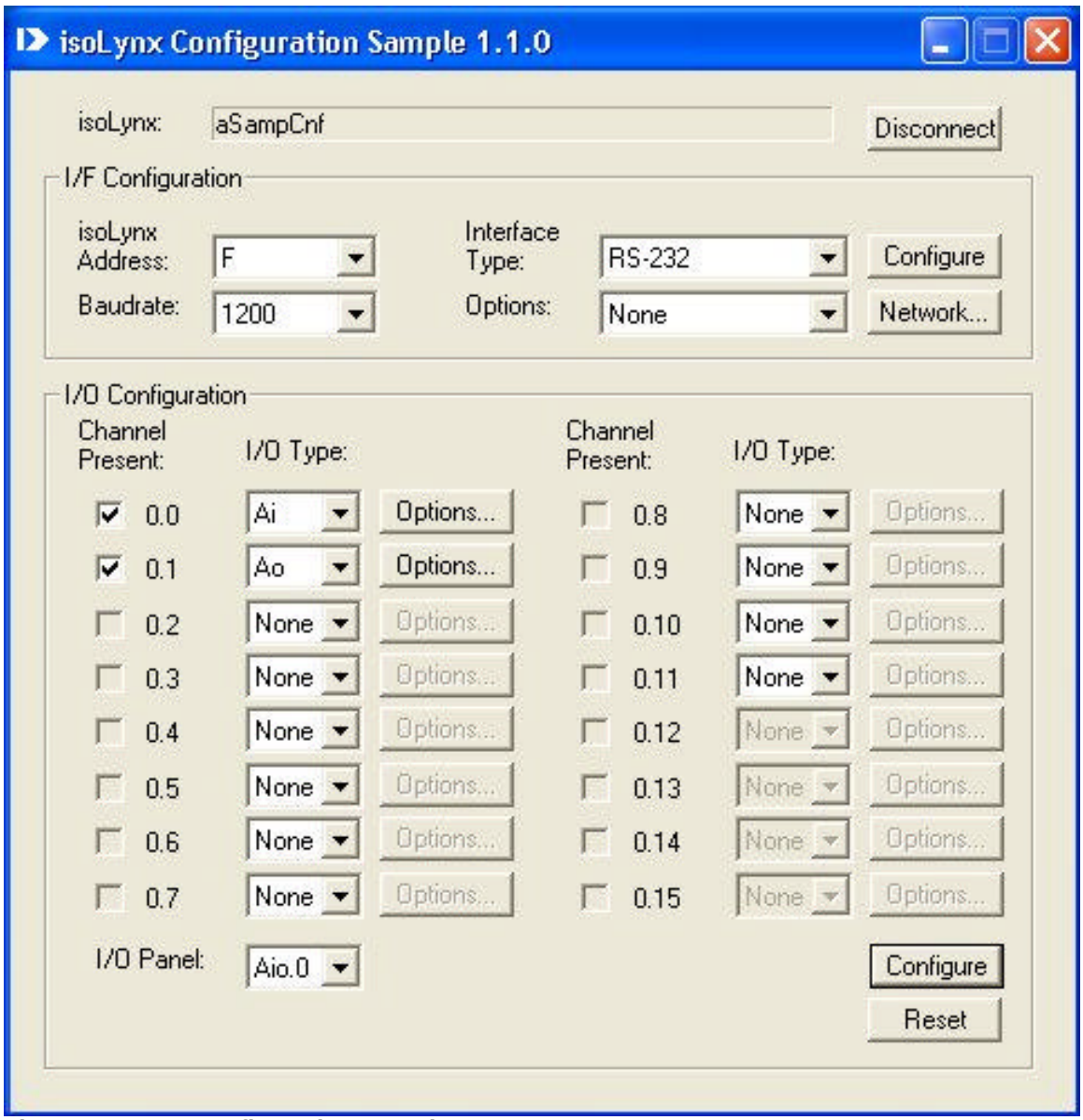


Figure 3.5.3.1 – Configuration Sample

- Change the communication interface by selecting new choices from the drop down list boxes in the **`I/F Configuration`** section of the window. When done selecting choices, click the **`Configure`** button. (Note that there are two **`Configure`** buttons. Click the one in the **`I/F Configuration`** section.) Changing the communication interface updates the isoLynx system's

communication settings and writes the new communication parameters to the script file.

WARNING: If you change the communication interface to one your host system does not support (for example, RS-485 2-wire) you will not be able to establish a connection with the isoLynx system again! Refer to the isoLynx ™ Hardware User Manual for information on restoring factory default communication settings to the isoLynx system.

- To configure I/O channels, the panel the channels reside on must be selected first. Select the I/O panel to configure by making an appropriate selection in the **I/O Panel** drop-down list box.

  Aio:0 is the isoLynx Analog I/O Base Unit and has twelve channels available for configuration.

  Aio1 – Aio3 refer to Analog I/O Expansion Backpanels that may be attached to the base system.

  Dio0 – Dio7 refer to isoLynx Digital I/O Backpanels that may be attached to the base system.

  See the isoLynx™ Hardware User Manual for more information on attaching analog and digital expansion panels to the base unit.

  After the panel is selected, I/O channels can be configured by selecting one of the following from the **I/O Type** drop-down list boxes for each channel: None (empty channel), Ai (analog input), Ao (analog output), Di (digital input), or Do (digital output). The Ai and Ao choices are only available if you selected an analog I/O panel. Likewise, the Di and Do choices are only available if you selected a digital I/O panel.

  Once the channels are configured to your satisfaction, click the **Configure** button. (Note that there are two **Configure** buttons. Click the one in the **I/O Configuration** section.)

- Channel options may be modified on a channel-by-channel basis by clicking the **Options…** button next to the corresponding channel. If the channel is an analog input, clicking the **Options…** button will open a dialog box that will allow the channel's sampling weight to be modified. This is the value used in calculating the channel's running average. If the channel is an analog output, clicking the **Options…** button will bring up a dialog box that will allow the channel's default output to be updated. This is the output value the channel will be initialized with on device reset or when the channel's configuration is changed to Output. (TODO: Channel options are not implemented for digital I/O panels.)

3.5.4 Input Sample

Once connected, the Input sample may be used to read channels that have been configured as inputs.

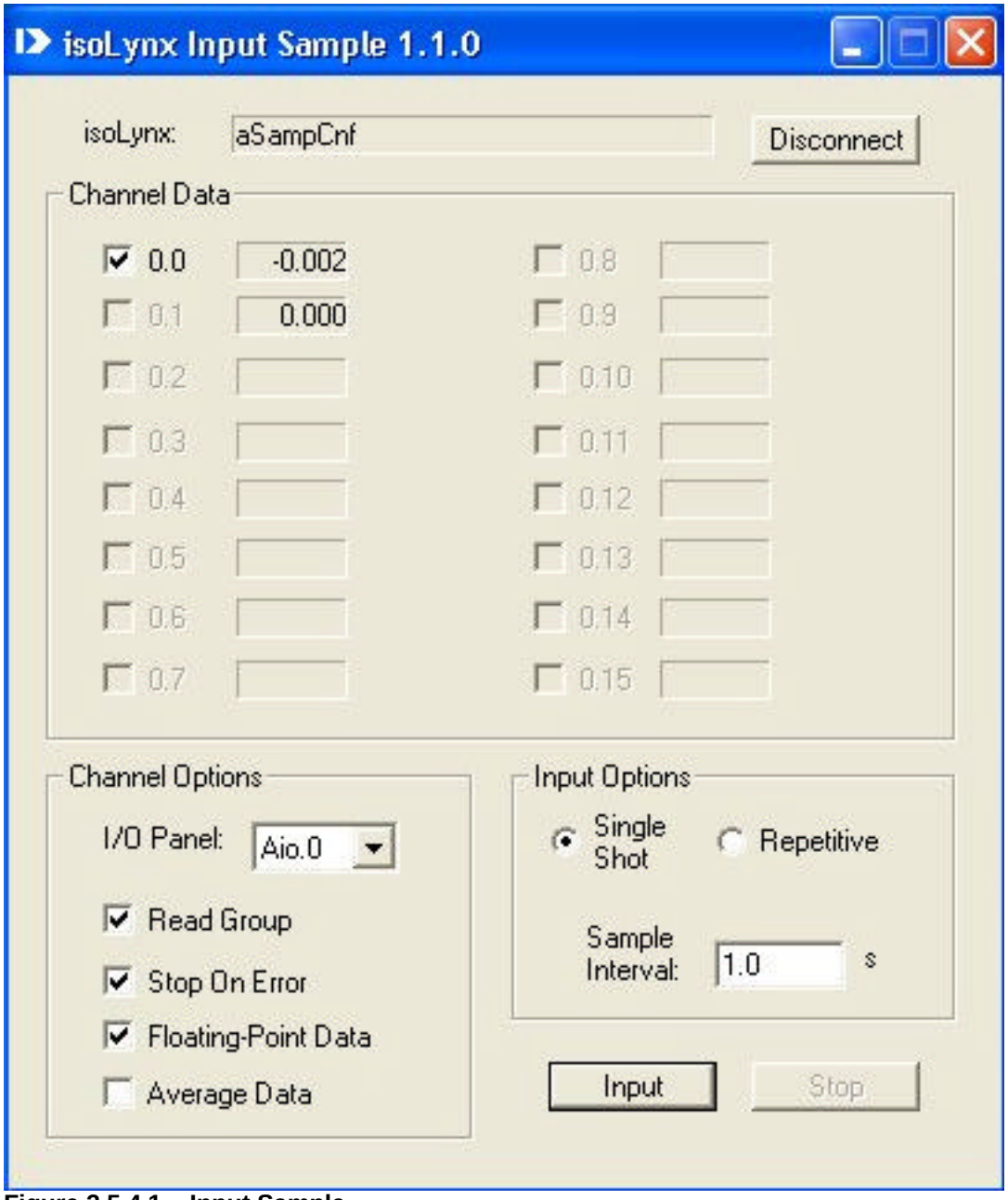


Figure 3.5.4.1 – Input Sample

First select the I/O panel to read from by making a selection from the **`I/O Panel`** drop-down list box. Then, simply check the appropriate boxes next to the

channel numbers you would like to read from. Be sure to read only from channels that have been configured as inputs. Otherwise, an error message will result. Other options are described below.

- Check the **Read Group** box to read all selected channels with one isoLynx command. Otherwise, the sample will read each selected channel separately, which is a much slower operation.

- Check the **Stop On Error** box to have the Input Sample quit reading data when it encounters an error. Otherwise, the program will log the error (to LYNXLIB.LOG) and keep reading data from the isoLynx system.

- Check the **Floating-Pt Data** box to have the Input Sample convert the 16-bit counts read from a channel to a floating point number. Uncheck the box to read raw counts.

- Check the **Average Data** box to read each channel's running average. Uncheck the box to read each channel's most recent sample.

- To read one time only, click the **Single Shot** radio button. To read continuously, select the **Repetitive** option and choose a sampling interval. Although you can choose smaller values, the smallest sampling interval you can obtain with the sample is approximately 50 ms. (50 ms is the resolution of the software timer used by the sample.)

Once all options have been selected, click the **Input** button. The data will be displayed in the channel data windows next to the channel numbers. If you chose the **Repetitive** option, the Input sample will continuously read data from the isoLynx system until the **Stop** button is clicked (or, if the **Stop On Error** option was selected, until an error occurs).

3.5.5 Output Sample

Once connected, the Output sample may be used to write data to channels that were configured as outputs.

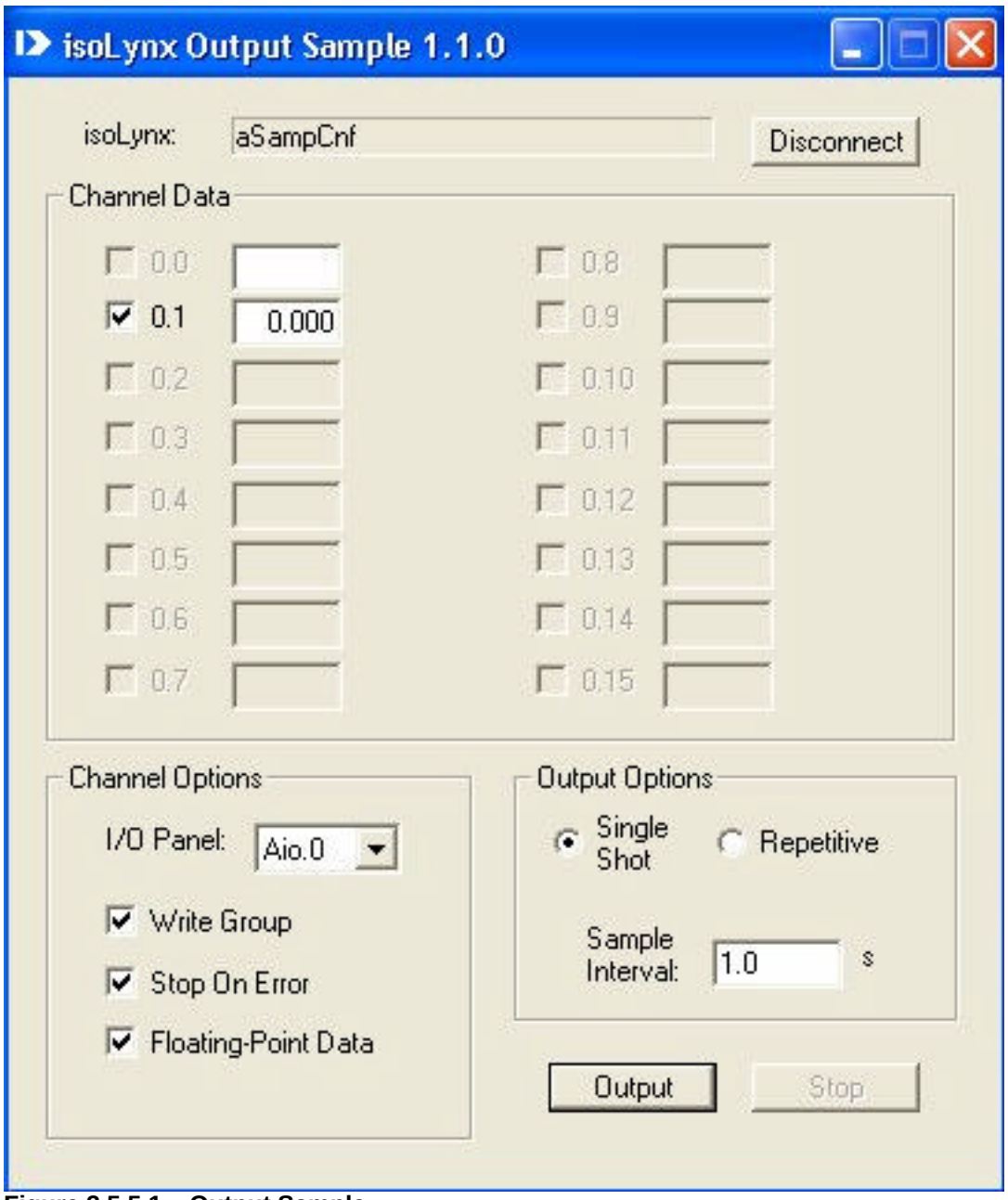


Figure 3.5.5.1 – Output Sample

First select the panel to write to by making a selection from the **`I/O Panel`** drop-down list box. Then, simply check the appropriate selection boxes and enter data to write into the channel data windows next to the appropriate channels. Be

sure to only select channels that have been configured as outputs. Otherwise, an error message will result. Other options are described below.

- Check the **Write Group** box to write all selected channels with one command. Otherwise, the sample will write each selected channel separately, which is a much slower operation.

- Check the **Stop On Error** box to have the Output Sample quit writing data when it encounters an error. Otherwise, the sample will log the error (to LYNXLIB.LOG) and continue writing data to the isoLynx system.

- Check the **Floating-Pt Data** box to enter data to write in floating point format (from -10.000 to 10.0000). Otherwise, the data entered into the channel data windows will have to be in 4-digit hexadecimal format (from 0000 to FFFF).

- To write one time only, click the **Single Shot** radio button. To write data continuously, select the **Repetitive** option and choose a desired interval. The interval tells the Sample how often to send a write command to the isoLynx. Although you can choose smaller values, the smallest write interval you can obtain with the sample is approximately 50 ms. (50 ms is the resolution of the software timer used by the sample.)

Once all options have been selected, click the **Output** button. The current contents of the channel data windows will be written to the selected channels. If the **Repetitive** option was chosen, the Output sample will continuously write the data it finds in the channel data windows. This data may be updated by the user at any time. Data will continue to be written until the **Stop** button is clicked (or, if the **Stop On Error** option was selected, until an error occurs).

4 Utilities

The previous section described how to use the isoLynx Data Acquisition Library and how to build custom applications with Visual Basic or Visual C++. This section describes how to use any higher-level utilities to build custom applications that utilize the isoLynx Library.

4.1 LabVIEW

A library of LabVIEW Virtual Instruments (VIs) is included on the installation CD. (`CD:\SAMPLES\LV\LYNXLIB.LLB`). These VIs are described in the following sections.

4.1.1 LabVIEW Virtual Instruments

This section documents the VIs included in the LYNXLIB.LLB library. These VIs utilize the isoLynx library by making calls to the library API functions. Please reference Section 3.3, isoLynx API for further details.

4.1.1.1 Open Library.VI

Opens the isoLynx Data Acquisition Library. This VI must run successfully before any other library VI's are executed. This VI calls the *dLibOpen* API function.

Connector Pane

Controls and Indicators

enable logging (T) is set to enable logging of error information to a logfile (LYNXLIB.LOG).

error in (no error) is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI passes the value of the **error in** cluster to **error out**.

- **status** is TRUE if an error occurred prior to the execution of this VI. If **status** is TRUE, the VI does not do anything other than pass **error in** information to **error out**.
- **code** is the error code number identifying an error. A value of 0 indicates no error, a negative value indicates a warning, and a positive value is a fatal error.
- **source** identifies the name of the VI that produced the error.

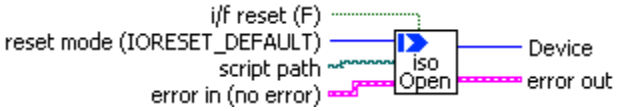
error out is a cluster that contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

- **status** is TRUE if an error occurred sometime prior to or during the execution of this VI. See **code** and **source** for more information.
- **code** is the error code number identifying an error. A value of 0 indicates no error, a negative value indicates a warning, and a positive value is a fatal error.
- **source** identifies the name of the VI that produced the error.

4.1.1.2 Open isoLynx Dev.VI

Establishes a communication session with a single isoLynx system. The Open Library VI must run to completion before this VI is called. This VI must successfully complete before several other isoLynx Data Acquisition VIs may be called (i.e. Read Data.vi, Write Data.vi, etc.). This VI calls the *dComOpen* and *dIoOpen* API functions.

Connector Pane



Controls and Indicators

reset mode (IORESET_DEFAULT) specifies the **reset mode** and how to deal with inconsistencies between the channel configuration discovered on the isoLynx system and that described by the supplied script file.

Select IORESET_NONE to not issue a reset as part of the open procedure. An appropriate warning is returned if a mismatch between the isoLynx and script file's channel configurations is discovered.

Select IORESET_DEFAULT to issue a standard reset to the isoLynx system (and all attached panels) as part of the open procedure. As with IORESET_NONE, an appropriate warning is returned if a mismatch between the isoLynx and script file's channel configurations is discovered.

Select IORESET_FACTORY to clear the isoLynx and script file's channel configurations (i.e. no channels configured).

Select IORESET_SCRIPTCONF to clear the isoLynx channel configuration, then duplicate on the isoLynx the channel configuration described in the script file. This is a handy way to configure the isoLynx channels with one command.

Select IORESET_SCRIPTSYNC to clear the script file's channel configuration, then duplicate in the script file the channel configuration discovered on the isoLynx system.

script path is the path to a script file. The script file contains communication settings and channel configuration information for an isoLynx system. See the isoLynx™ Software User Manual for more information on script files.

i/f reset (F) is set to reset the [Isolynx] section of the supplied script file to that defined in ISOLYNX.INI. Do this after the isoLynx system has had its communication settings reset to factory defaults (RS-232 @ 9600 baud). (See the isoLynx Hardware User Manual for instructions on resetting the isoLynx system's communication settings.) Setting the **i/f reset** flag synchronizes the [Isolynx] section of the script file with the isoLynx system.

error in (no error) is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI passes the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred prior to the execution of this VI. If **status** is TRUE, the VI does not do anything other than pass **error in** information to **error out**.

code is the error code number identifying an error. A value of 0 indicates no

error, a negative value indicates a warning, and a positive value is a fatal error.

source identifies the name of the VI that produced the error.

Device is a handle to an opened isoLynx system. This value is required as an input by several other isoLynx Data Acquisition VI's.

error out is a cluster that contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred sometime prior to or during the execution of this VI. See **code** and **source** for more information.

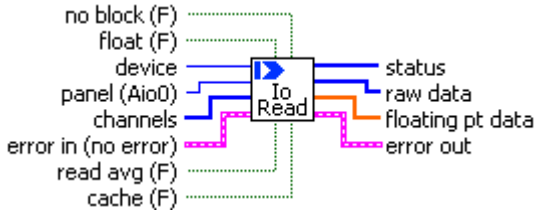
code is the error code number identifying an error. A value of 0 indicates no error, a negative value indicates a warning, and a positive value is a fatal error.

source identifies the name of the VI that produced the error.

4.1.1.3 Read Data.VI

Performs a single group read of one or more isoLynx I/O channels. This VI calls the *dIoRead* API function.

Connector Pane



Controls and Indicators

device is the handle to an opened isoLynx system.

panel (Aio0) is the panel address of the I/O panel to read from:

Aio0 indicates the isoLynx Analog I/O Base Unit.

Aio1-3 indicate Analog I/O Expansion Backpanels.

Dio0-7 indicate Digital I/O Backpanels.

channels is an array of channel numbers of each I/O channel to read. Valid values for each channel are 0-15.

no block (F) is set to true to not wait for results from the isoLynx system before returning to the caller. This VI may then be used to poll for completion. This is an advanced mode of operation and should be used only when the normal mode of operation (blocking) is insufficient.

float (F) is set to true to read floating point data instead of raw data. A formula is applied to the 16-bit raw data value read from the isoLynx system to derive a floating-point value. The formula used is:

floating point data = (raw data * gain) + offset

A channel's gain and offset parameters are set when the channel is configured.

cache (F) is set to true to read the last data written to or read from the isoLynx device. If true, no command is sent to the isoLynx system. Instead, the most recent values written to or read from the device are returned.

Note that setting this flag is the only way to read the most recent value written to an output channel.

read avg (F) is set to true to read an I/O channel's running average instead of its current value.

error in (no error) is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI passes the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred prior to the execution of this VI. If **status** is TRUE, the VI does not do anything other than pass **error in** information to **error out**.

code is the error code number identifying an error. A value of 0 indicates no error, a negative value indicates a warning, and a positive value is a fatal error.

source identifies the name of the VI that produced the error.

status contains read status for each channel (in the form of an error code). The size of this array will be equal to the size of the **channels** array. Status at a given index belongs to the channel specified at the same index in the **channels** array. For example, if the channels array contains {0, 5, 11}, then **status**[0] contains status from channel 0, **status**[1] contains status from channel 5, and **status**[2] contains status from channel 11.

raw data If **float** was set to true, this array will be empty. Otherwise, this array contains the raw data read from each of the channels specified in the **channels** array. The size of this array will be equal to the size of the **channels** array.

Data at a given index belongs to the channel specified at the same index in the **channels** array. For example, if the **channels** array contains {0, 5, 11}, then **raw data**[0] contains data read from channel 0, **raw data**[1] contains data read from channel 5, and **raw data**[2] contains data read from channel 11.

floating pt data If **float** was set to false, this array will be empty. Otherwise, this array contains the floating point data read from each of the channels specified in the **channels** array. The size of this array will be equal to the size of the **channels** array.

All elements of this array will have had the following formula applied to the raw 16-bit data value read from the I/O channel. *gain* and *offset* are channel specific parameters set when the channel was configured.

*floating pt data = (raw data * gain) + offset*

Data at a given index belongs to the channel specified at the same index in the **channels** array. For example, if the **channels** array contains {0, 5, 11}, then **floating pt data**[0] contains data read from channel 0, **floating pt data**[1] contains data read from channel 5, and **floating pt data**[2] contains data read from channel 11.

error out is a cluster that contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred sometime prior to or during the execution of this VI. See **code** and **source** for more information.

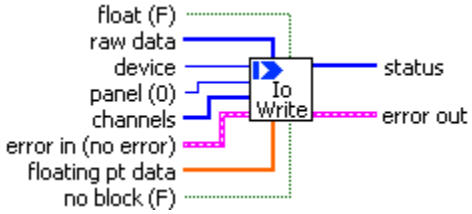
code is the error code number identifying an error. A value of 0 indicates no error, a negative value indicates a warning, and a positive value is a fatal error.

source identifies the name of the VI that produced the error.

4.1.1.4 Write Data.VI

Performs a single group write to one or more isoLynx output channels. This VI calls the *dIoWrite* API function.

Connector Pane



Controls and Indicators

device is the handle to an opened isoLynx system.

panel (0) is the panel address of the I/O panel to read from:

Aio0 indicates the isoLynx Analog I/O Base Unit.

Aio1-3 indicate Analog I/O Expansion Backpanels.

Dio0-7 indicate Digital I/O Backpanels.

channels is an array of channel numbers of each output channel to write. Valid values for each channel are 0-15.

float (F) is set to true to write floating-point data instead of raw data. A formula is applied to the floating-point data to convert it to a 16-bit raw data value to write to the isoLynx system. The formula used is:

raw data = (floating point data - offset) / gain

A channel's *gain* and *offset* parameters are set when the channel is configured.

no block (F) is set to true to not wait for results from the isoLynx system before returning to the caller. This VI may then be used to poll for completion. This is an advanced mode of operation and should be used only when the normal mode of operation (blocking) is insufficient.

raw data If **float** is set to true, this array should be empty. Otherwise, this array should contain the raw data to write to each of the channels specified in the **channels** array. Valid values are 0-FFFF hexadecimal.

The size of this array should be equal to the size of the **channels** array. Data at a given index will be written to the channel specified at the same index in the **channels** array. For example, if the **channels** array contains {0, 5, 11}, then **floating pt data**[0] contains data to write to channel 0, **floating pt data**[1] contains data to write to channel 5, and **floating pt data**[2] contains data to write to channel 11.

floating pt data If **float** is set to false, this array should be empty. Otherwise, this array should contain the floating point data to write (before adjustment for *gain* and *offset*) to each of the channels specified in the **channels** array.

Each element of this array will have the following formula applied to it, using the corresponding channel's *gain* and *offset* parameters, before it is written to the channel:

raw data = (floating pt data - offset) / gain

If the resulting number is greater than FFFF hexadecimal, it will be truncated by the library.

The size of this array should be equal to the size of the **channels** array. Data at a given index will be written to the channel specified at the same index in the **channels** array. For example, if the **channels** array contains {0, 5, 11}, then **floating pt data**[0] contains data to write to channel 0, **floating pt data**[1] contains data to write to channel 5, and **floating pt data**[2] contains data to write to channel 11.

error in (no error) is a cluster that describes error conditions occurring before the VI executes. If an error has already occurred, the VI passes the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred prior to the execution of this VI. If **status** is TRUE, the VI does not do anything other than pass **error in** information to **error out**.

code is the error code number identifying an error. A value of 0 indicates no error, a negative value indicates a warning, and a positive value is a fatal error.

source identifies the name of the VI that produced the error.

status contains write status for each channel (in the form of an error code). The size of this array will be equal to the size of the **channels** array. Status at a given index belongs to the channel specified at the same index in the **channels** array. For example, if the channels array contains {0, 5, 11}, then **status**[0] contains status from channel 0, **status**[1] contains status from channel 5, and **status**[2] contains status from channel 11.

error out is a cluster that contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

status is TRUE if an error occurred sometime prior to or during the execution of this VI. See **code** and **source** for more information.

code is the error code number identifying an error. A value of 0 indicates no error, a negative value indicates a warning, and a positive value is a fatal error.

source identifies the name of the VI that produced the error.

4.1.1.5 Close isoLynx Dev.VI

Terminates the communication session with an isoLynx system. This VI must run before the Close Library VI does. This VI calls the *dIoClose* API function.

Connector Pane

Controls and Indicators

device is the handle to an opened isoLynx system.

error in (no error) is a cluster that describes error conditions occurring before the VI executes.

This VI will attempt to execute regardless of any previous error. If successful, the **error in** cluster (which may indicate no error) is passed to **error out**. If unsuccessful, a new error cluster is built and passed to **error out**.

- **status** is TRUE if an error occurred prior to the execution of this VI.
- **code** is the error code number identifying an error. A value of 0 indicates no error, a negative value indicates a warning, and a positive value is a fatal error.
- **source** identifies the name of the VI that produced the error.

error out is a cluster containing error information. If this VI was successful, **error out** is the same as **error in** (which might be no error). If this VI was not successful, **error out** indicates the type of error.

- **status** is TRUE if an error occurred during or sometime prior to execution of this VI. See the **code** and **source** for more information.
- **code** is the error code number identifying an error. A value of 0 indicates no error, a negative value indicates a warning, and a positive value is a fatal error.
- **source** identifies the name of the VI that produced the error.

4.1.1.6 Close Library.VI

Closes the isoLynx Data Acquisition Library. This VI should be the last isoLynx Data Acquisition VI to execute. This VI calls the *dLibClose* API function.

Connector Pane

Controls and Indicators

error in (no error) is a cluster that describes error conditions occurring before the VI executes.

This VI will attempt to execute unless an error occurred in the Close isoLynx Dev VI. The library shouldn't be closed if the isoLynx device failed to close.

status is TRUE if an error occurred prior to the execution of this VI.

code is the error code number identifying an error. A value of 0 indicates no error, a negative value indicates a warning, and a positive value is a fatal error.

source identifies the name of the VI that produced the error.

error out is a cluster that contains error information. In this VI, **error out** is always the same as **error in**.

status is TRUE if an error occurred sometime prior to the execution of this VI. See the **code** and **source** for more information.

code is the error code number identifying an error. A value of 0 indicates no error, a negative value indicates a warning, and a positive value is a fatal error.

source identifies the name of the VI that produced the error.

4.1.1.7 Decode Error.VI

Decode Error.vi

Given an error code, decodes the error by looking for a corresponding error code in any *.ERR files. If found, the textual description in the *.ERR file is returned. This VI calls the *dLibError* API function.

Connector Pane

Controls and Indicators

error in (no error) is a cluster describing an error condition to decode.

- **status** is TRUE if an error occurred prior to the execution of this VI.
- **code** is the error code number identifying an error. A value of 0 indicates no error, a negative value indicates a warning, and a positive value is a fatal error.
- **source** identifies the name of the VI that produced the error.

decoded error is a textual description of the error code.

error out is a cluster containing error information. For this VI, **error out** is always the same as **error in**.

- **status** is TRUE if an error occurred sometime prior to the execution of this VI. See **code** and **source** for more information.
- **code** is the error code number identifying an error. A value of 0 indicates no error, a negative value indicates a warning, and a positive value is a fatal error.
- **source** identifies the name of the VI that produced the error.

4.1.2 Sample LabVIEW Applications

A set of sample LabVIEW applications is provided to demonstrate usage of the isoLynx VI Library (LYNXLIB.LLB). The following section describes how to install and run the LabVIEW samples.

4.1.2.1 Running the LabVIEW Samples

The following steps describe how to run the sample applications. It is assumed that you already have a recent version of LabVIEW installed (version 6.1 or later) on your host system. You will be unable to use the samples if LabVIEW is not installed.

It is also assumed that you already have a powered-on isoLynx system connected to the host system. If this is not the case, see the isoLynx™ Hardware User Guide for instructions on installing an isoLynx system.

1. If it doesn't already exist, create the C:\PROGRAM FILES\DATAFORTH\ISOLYNX folder on your local hard drive. From the installation CD-ROM, copy the CD:\SAMPLES\LV\LVDEMO folder and all of its contents to the folder you just created.

2. The channels need to be configured before using the LabVIEW samples. Run CONFIG.EXE from the LVDEMO directory on your local hard drive. Connect to the isoLynx system by clicking the **Connect…** button. Verify the default script file (ASAMPCNF.DAT) is selected in the resulting dialog box. Click **Connect** in the dialog box to establish a connection.

   Note: The communication settings in the script file must match the communication settings on the isoLynx for a connection attempt to be successful. If either have been changed, you may need to reset the communication settings to the factory defaults. Do this for the script file by checking the **Use factory default COM settings checkbox**. See the isoLynx™ Hardware User Guide for instructions on resetting the isoLynx communication settings.

3. Configure the channel configuration and communication settings to your liking. See Section 3.5.3 for detailed instructions on how to do so.

4. Open a sample VI from the LV_DEMO directory on your local hard drive. Run it by clicking the arrow on the toolbar in the upper-left corner of the window.

4.1.3 Creating LabVIEW Applications

Developing your own custom data acquisition applications with LabVIEW is easy. Simply use the VIs provided in the isoLynx VI library (LYNXLIB.LLB) as building blocks in your own application. The sample VIs are a good example of this.

For more advanced applications, you might consider developing some lower-level VIs that access the isoLynx Data Acquisition Library directly. This may be done by using LabVIEW Code Interface Nodes (CINs) in your VI block diagrams. A Code Interface Node allows external code to be called from LabVIEW. See current LabVIEW documentation for detailed information on CINs and other methods of calling external code from LabVIEW.

Appendix A Troubleshooting Guidelines

A.1 isoLynx™ Controller "A/D" LED Blink Patterns

The following LED blink patterns identify various correct and/or erroneous operational modes of the isoLynx bootup, self-test, and continuous modes. Whenever you encounter any of the erroneous mode blink patterns, remember to check the hardware setup and connections and reference the isoLynx™ Hardware User Manual.

Equal ON/OFF	isoLynx booted normally, fully operational, awaiting commands.
Long Blink	Long ON, Short OFF. Communication timeout. isoLynx has not received a command from the host within the user set Watchdog timeout period.
Short Blink	Short ON, Long OFF. I/O Signal Converter Board alert.
2 Short Blinks	Short ON, Short OFF, Short ON, Long OFF. Processor Board alert.
Full OFF	No power, LED circuit failed OFF, isoLynx firmware hung when LED was OFF.
Full ON	LED circuit failed ON, isoLynx firmware hung when LED was ON.

A.2 If the isoLynx™ Does Not Communicate or Sends Garbled Data From Any Interface

If the isoLynx Analog I/O Base Unit or Digital I/O Backpanel boots correctly but has unknown communication parameters, the communication parameters can be reset to a known state.

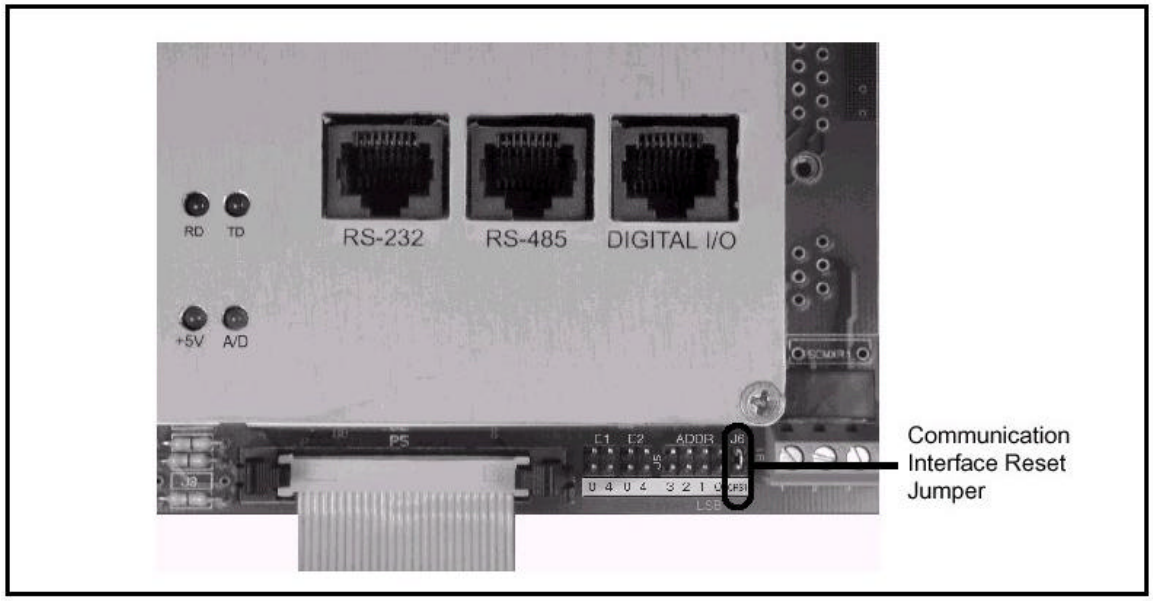


Figure A.2-1

* Open the Communication Interface Reset Jumper momentarily with the mini-link shunt jumper provided on header J6. The shunt jumper must be re-installed over both pins and left there for the reset to complete and for continuous operation to begin.

* The isoLynx will be triggered to start a boot-up sequence. Once it has finished booting up, you may communicate with it through the RS-232 port at 9.6K bits per second (bps) (Baud).

A.3 Selected Error Codes and Some of Their Less Obvious Causes.

* If error code 02, Checksum Error, persists, beyond the obvious miscalculated checksum, it could be an indication of characters being dropped from the serial data stream. Recheck all communication cables and connectors for intermittent operation by wiggling cables and connectors. It could be caused by contamination or oxidation on connector pins and sockets unplug and replug all connectors. If the problem persists, replace suspected cables and connectors. Lastly, suspect the data communications electronic components, the communications ports in the host computer, in the isoLynx Analog I/O Base Unit, and/or the isoLynx Digital I/O Backpanel. Try another port in the host computer and/or try another isoLynx Analog I/O Base Unit or isoLynx Processor Board or isoLynx Digital I/O Backpanel. Reference the isoLynx™ Hardware User Manual.

* If error code 06, Communications Link Watchdog Time-Out Error, persists, this could be caused by a problem with the host computer communications link or by a problem with the isoLynx Digital I/O Backpanel communications link. This is similar to error code 02, Checksum Error, above except the communication drop-outs last long enough to trigger a watchdog time-out. The troubleshooting steps are the same as above, but in addition to the host connections, check the isoLynx Digital I/O Backpanel connections. For this case also check the integrity of the power sources, cables, and connections of the host computer and the isoLynx Digital I/O Backpanel. Reference the isoLynx™ Hardware User Manual.

* If error code 09, Invalid Module Type Error, persists, cross-check the I/O configuration stored in the isoLynx against the actual module types and their locations in your isoLynx system hardware.

A.4 Unexpected Data Values and Some Possible Causes

* If unexpected data values which do not cause checksum errors occur, these could be caused by a problem with the host computer communications link or by a problem with the isoLynx Digital I/O Backpanel communications link. Recheck all communication cables and connectors for intermittent operation by wiggling cables and connectors. It could be caused by contamination or oxidation on connector pins and sockets unplug and replug all connectors. If the problem persists, replace suspected cables and connectors. Lastly, suspect the data communications electronic components, the communications ports in the host computer, in the isoLynx Analog I/O Base Unit, and/or the isoLynx Digital I/O Backpanel. Try another port in the host computer and/or try another isoLynx Analog I/O Base Unit or isoLynx Processor Board or isoLynx Digital I/O Backpanel. Reference the isoLynx™ Hardware User Manual.

Appendix B - ASCII Character Table

CHAR.	HEX	CHAR.	HEX	CHAR.	HEX	CHAR.	HEX
Ctrl-@	00	Space	20	@	40	`	60
Ctrl-A	01	!	21	A	41	a	61
Ctrl-B	02	“	22	B	42	b	62
Ctrl-C	03	#	23	C	43	c	63
Ctrl-D	04	$	24	D	44	d	64
Ctrl-E	05	%	25	E	45	e	65
Ctrl-F	06	&	26	F	46	f	66
Ctrl-G	07	’	27	G	47	g	67
Ctrl-H	08	(	28	H	48	h	68
Ctrl-I	09	)	29	I	49	i	69
Ctrl-J	0A	*	2A	J	4A	j	6A
Ctrl-K	0B	+	2B	K	4B	k	6B
Ctrl-L	0C	,	2C	L	4C	l	6C
Ctrl-M	0D	-	2D	M	4D	m	6D
Ctrl-N	0E	.	2E	N	4E	n	6E
Ctrl-O	0F	/	2F	O	4F	o	6F
Ctrl-P	10	0	30	P	50	p	70
Ctrl-Q	11	1	31	Q	51	q	71
Ctrl-R	12	2	32	R	52	r	72
Ctrl-S	13	3	33	S	53	s	73
Ctrl-T	14	4	34	T	54	t	74
Ctrl-U	15	5	35	U	55	u	75
Ctrl-V	16	6	36	V	56	v	76
Ctrl-W	17	7	37	W	57	w	77
Ctrl-X	18	8	38	X	58	x	78
Ctrl-Y	19	9	39	Y	59	y	79
Ctrl-Z	1A	:	3A	Z	5A	z	7A
Ctrl-[	1B	;	3B	[	5B	{	7B
Ctrl-\	1C	<	3C	\	5C	|	7C
Ctrl-]	1D	=	3D	]	5D	}	7D
Ctrl-^	1E	>	3E	^	5E	~	7E
Ctrl-_	1F	?	3F	__	5F	DEL	7F

Appendix C - Warranty, Disclaimers, Return/Repair Policy

WARRANTY

General. Seller warrants that its products furnished hereunder will, at the time of delivery, be free from defects in material and workmanship and will conform to Seller's applicable specifications or, if appropriate, to Buyer's specifications accepted in writing by Seller. SELLER'S OBLIGATION OR LIABILITY TO BUYER FOR PRODUCTS WHICH DO NOT CONFORM TO THE ABOVE STATED WARRANTY SHALL BE LIMITED TO SELLER, AT SELLER'S SOLE DISCRETION, EITHER REPAIRING, REPLACING, OR REFUNDING THE PURCHASE PRICE OF THE DEFECTIVE PRODUCT(S) PROVIDED THAT WRITTEN NOTICE OF SAID DEFECT IS RECEIVED BY SELLER WITHIN THE TIME PERIODS SET FORTH BELOW:

i. for all software products including licensed programs, thirty (30) days from date of initial delivery;

ii. for all hardware products including complete systems, one (1) year from date of initial delivery;

iii. for all special products, sixty (60) days from date of initial delivery; and

further, all products warranted hereunder for which Seller has received timely notice of nonconformance must be returned FOB Seller's plant within thirty (30) days after the expiration of the warranty periods set forth above.

The foregoing warranties shall not apply to any products which Seller determines have, by Buyer or otherwise, been subjected to operating and/or environmental conditions in excess of the maximum value established therefor in the applicable specifications, or any products that have been the subject of mishandling, misuse, misapplication, neglect, improper testing, repair, alteration or damage.

Limitation. THE PROVISIONS OF THE FOREGOING WARRANTIES EXTEND TO BUYER ONLY AND NOT TO BUYER'S CUSTOMERS OR USERS OF BUYER'S PRODUCTS AND ARE IN LIEU OF ANY OTHER WARRANTY, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL SELLER BE LIABLE FOR INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES. Seller's liability arising out of the production, sale or supply of products or their use or disposition, whether based upon warranty, contract, tort or otherwise, shall not exceed the actual purchase price paid by Buyer for Seller's products. Seller's liability for any claim of any kind shall in no case exceed the obligation or liability specified in this Warranty.

Technical Assistance. Seller's Warranty as hereinabove set forth shall not be enlarged, diminished or affected by, and no obligation or liability shall arise or grow out of, Seller's rendering of technical advice, facilities or service in connection with Buyer's order of the goods furnished hereunder.

Warranty Procedures. Buyer shall notify Seller of any products which it believes to be defective during the applicable warranty period and which are covered by the warranty set forth above. Buyer shall not return any products for any reason without the prior authorization of Seller and issuance of a Return Material Authorization number. After issuance of an RMA number, such products shall be promptly returned by Buyer (and in no event later than thirty (30) days after the warranty expiration date), transportation and insurance prepaid, to the Seller's designated facility for examination and testing. Seller shall either repair or replace any such products found to be so defective and promptly return such products to Buyer, transportation and insurance prepaid. Should Seller's examination and testing not disclose any defect covered by the foregoing warranty, Seller shall so advise Buyer and dispose of or return the products in accordance with Buyer's instructions and at Buyer's sole expense, and Buyer shall reimburse Seller for testing expenses incurred at Seller's then current repair rates.

Repair Warranty. Seller warrants its repair work and/or replacement parts for a period of ninety (90) days from receipt by Buyer of the repaired or replaced products or for the remainder of the warranty period for the initial delivery of such order as set forth above in paragraph a, whichever is greater.

Critical Applications. Certain applications using Seller's products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications"). SELLER'S PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS, SAFETY EQUIPMENT, NUCLEAR FACILITY APPLICATIONS OR OTHER CRITICAL APPLICATIONS WHERE MALFUNCTION OF THE PRODUCT CAN BE EXPECTED TO RESULT IN PERSONAL INJURY, DEATH OR SEVERE PROPERTY DAMAGE. BUYER USES OR SELLS SUCH PRODUCTS FOR USE IN SUCH CRITICAL APPLICATIONS AT BUYER'S OWN RISK AND AGREES TO DEFEND, INDEMNIFY AND HOLD HARMLESS SELLER FROM ANY AND ALL DAMAGES, CLAIMS, SUITS OR EXPENSE RESULTING FROM SUCH USE.

Static Sensitive. Seller ships all product in anit-static packages. Seller's Warranty as hereinabove set forth shall not cover warranty repair, replacement, or refund on product or devices damaged by static due to Buyer's failure to properly ground.

Return/Repair Policy

All warranty and repair requests should be directed to the Dataforth Customer Service Department at (520) 741-1404. If a product return is required, request a Return Material Authorization (RMA) number. You should be ready to provide the following information:

1. Complete product model number.
2. Product serial number.
3. Name, address, and telephone number of person returning product.
4. Special repair instructions.
5. Purchase order number for out-of-warranty repairs.

The product should be carefully packaged, making sure the RMA number appears on the outside of the package, and ship prepaid to:

Dataforth Corporation
3331 E. Hemisphere Loop
Tucson, AZ 85706 USA

The information provided herein is believed to be reliable; however, DATAFORTH assumes no responsibility for inaccuracies or omissions. DATAFORTH assumes no responsibility for the use of this information, and all use of such information shall be entirely at the user's own risk. Application information is intended as suggestions for possible use of the products and not as explicit performance in a specific application. Prices and specifications are subject to change without notice. No patent rights or licenses to any of the circuits described herein are implied or granted to any third party. DATAFORTH does not authorize or warrant any DATAFORTH product for use in life support devices and/or systems.